Hochschule RheinMain
Fachbereich Design Informatik Medien
Studiengang Allgemeine Informatik

Bachelor-Thesis
Zur Erlangung des akademischen Grades
Bachelor of Science B.Sc.

Untersuchung von Interaktionskonzepten in 3D-Umgebungen auf Smartphones mit dem Android Betriebssystem.

vorgelegt von Kai Groetenhardt
Herrngartenstraße 16
65185 Wiesbaden

am 15.06.2011

Referent: Prof. Dr. Ralf Dörner
Korreferent: Prof. Dr. Christoph Schulz

Erklärung gem. ABPO, Ziff. 6.4.3

Ich versichere, dass ich die Bachelor-Thesis selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Ort, Datum

Kai Groetenhardt

Hiermit erkläre ich mein Einverständnis mit den im folgenden aufgeführten Verbreitungsformen dieser Bachelor-Thesis:

Verbreitungsform	**ja**	**nein**
Einstellung der Arbeit in die Bibliothek der Hochschule RheinMain	x	
Veröffentlichung des Titels der Arbeit im Internet	x	
Veröffentlichung der Arbeit im Internet	x	

Ort, Datum

Kai Groetenhardt

Inhaltsverzeichnis

Abbildungsverzeichnis

Codeverzeichnis

Aufgabenstellung

Im Rahmen der Bachelor Thesis soll untersucht werden, welche interaktive Steuerungsmöglichkeiten bei einem Smartphone für die virtuelle Kamera und welche Manipulationsmöglichkeiten für Objekte in einem drei-dimensionalem Raum denkbar sind und dabei auch eine gute Benutzerakzeptanz haben. Als Modell für ein Smartphone dient dazu ein Android Smartphone in der aktuellen Version 2.2.

Die zu untersuchenden Steuerungen und Manipulationen umfassen, unter anderem, das Multitouch-Display und verschiedene Sensoren, die dem Smartphone zur Verfügung stehen. Dabei ist zu beachten, dass für ein Smartphone andere Interaktionen zur Verfügung stehen als für einen regulären PC.

Im Rahmen dieser Untersuchung sollten mehrere Interaktionskonzepte entwickelt und davon mindestens zwei implementiert werden, welche unter Angabe von Gründen zu wählen sind. Eine Implementierung aller Konzepte wird nicht angestrebt, da dies im Rahmen der Arbeit nicht leistbar ist. Die Konzepte, die nicht zur Implementierung ausgewählt wurden, werden theoretisch beschrieben. Vor der Formulierung eigener Konzepte sind bisherige Arbeiten beispielhaft zu untersuchen und zu bewerten.

Die dreidimensionale Testumgebung für das jeweilige Interaktionskonzept wird in Form eines simplen Spiels erstellt. Dies bedeutet, dass zwei Spiele auf Grundlage unterschiedlicher Konzepte erstellt werden, welche sich in ihrer Schwierigkeit nicht unterscheiden sollten, um die Vergleichbarkeit aufrecht zu erhalten. Der Schwerpunkt der Spiele liegt in der Interaktion in der dreidimensionalen Umgebung. Denkbar wäre beispielsweise folgendes Spiel: Verschiedene Objekte müssen in einem Raum von einem Ort zu einem anderen transportiert werden, wobei alle drei Dimensionen eine Rolle spielen. Zum Beispiel müssen Objekte vom Boden aufgehoben und in eine Kiste in auf einer erhöhten Ebene befördert werden.

Wurden die Interaktionskonzepte umgesetzt, sollen diese im Rahmen eines Usertests miteinander verglichen werden. Zur Bewertung wird den Testusern ein Fragebogen vorgelegt, mit dem Aussagen über die Konzepte bewertet werden, um daraus statistische Aussagen ermitteln zu können. Gesamtziel der Usertests ist es Aussagen zu treffen, welches Interaktionskonzept von den Benutzern präferiert wird.

1 Einleitung

1.1 Motivation

In den letzten Jahren haben Multitouch-Displays immer mehr an Bedeutung gewonnen, insbesondere seitdem auch in Smartphones Multitouch-Displays verbaut werden. Die einfache Eingabe direkt am Display ohne weitere Geräte wie Maus und Tastatur erleichtern die Bedienung vor allem unterwegs enorm **[Bre10]** .

Zu beobachten ist, dass sich in 2D Applikationen einige Gesten durchgesetzt haben, die für entsprechende Aufgaben auf unterschiedlichsten Plattformen verwendet werden. Beispielsweis wird in Bildergalerien (z.B. auf dem iPhone oder Smartphones mit Android) meistens die Zoom-Geste zum Vergrößern oder Verkleinern der Bilder angewendet. Für die Zoom-Geste werden zwei Finger auf einem Bild platziert und auseinander geschoben, um das Bild zu vergrößern, bzw. zusammen geschoben, um es zu verkleinern.

Bei 3D Applikationen kann man das nicht beobachten, dort sind mehrere Gesten verbreitet, um bestimmte Aktionen auszuführen. Beispielsweise wird in einigen Applikationen das Drehen und Neigen der virtuellen Kamera mit einem Finger durchgeführt, in anderen mit zwei Fingern. Weitere Beispiele können aus dem Kapitel **4 Beispielhafte Untersuchung vorhandener Interaktionskonzepte** entnommen werden. Anders gesagt, ist hier kein durchgehender Standard festzustellen.

Da sich noch kein fester Standard für die Navigation im dreidimensionalen Raum für Smartphones etabliert hat, bietet sich die interessante Möglichkeit neuartige Navigationsmöglichkeiten zu entwickeln und untereinander zu vergleichen, um festzustellen welches die beste Benutzerfreundlichkeit ergibt.

1.2 Herangehensweise

Zu Anfang werden einige bereits vorhandene Interaktionskonzepte untersucht und beurteilt. Dabei sollte festgestellt werden für welche Arten von Interaktionen sich diese Konzepte besonders gut eignen und inwiefern einzelne Elemente der Konzepte wiederverwendet werden können.
Im zweiten Schritt werden eigene Interaktionskonzepte entwickelt, welche auf den bereits vorhandenen Konzepten aufbauen können.
Aus den entwickelten Konzepten werden zwei für die praktische Umsetzung ausgewählt.
Für die praktische Umsetzung wird eine Testumgebung in Form eines kleinen Spieles erstellt.

Nach der Entwicklung werden die Konzepte in Form eines Usertest miteinander verglichen, primär um festzustellen welches auf eine bessere Userakzeptanz stößt.
Zum Abschluss könnten aus den Ergebnissen des Tests Schlüsse gezogen werden, was an den Konzepten verbessert werden könnte, bzw. welches Konzept verbessert werden sollte.

1.3 Ziel

In dieser Arbeit werden zwei möglichst verschiedene Interaktionskonzepte für 3D-Umgebungen auf Android-Smartphones entwickelt, welche mit Hilfe einer virtuellen Testumgebung in Form eines Spiels getestet werden.
Der Test wird durch freiwillige Testuser durchgeführt, die die Konzepte mit Hilfe eines Fragebogens bewerten. Aufgrund dieser Bewertungen kann evaluiert werden, welches der Konzepte in die richtige Richtung zur besten Userakzeptanz geht.
Auf Grundlage dessen ist es eventuell möglich eine Tendenz abzulesen, in welche Richtung zukünftige Konzepte entwickelt werden müssen.

1.4 Inhaltsangabe

Nach der Einleitung folgt in Kapitel 2 die Problemanalyse, welche das grundlegende Problem beschreibt und die Ist-Situation kurz darlegt.
In Kapitel 3 folgen die Grundlagen, welche dem tieferen Verständnis diese Arbeit dienen.
Im darauf folgenden Kapitel ist die beispielhafte Untersuchung vorhandener Interaktionskonzepte zu finden. In diesem Kapitel werden einige Konzepte kurz vorgestellt und beurteilt. Zudem wird erläutert welche Gestern der Konzepte Einzug in eigene Konzepte finden könnten.
In dem darauf folgenden Abschnitt werden die eigenen Interaktionskonzepte beschrieben, die im Laufe der Thesis erarbeitet wurden.
Kapitel 6 zeigt welche der Konzepte zur Implementierung ausgewählt wurden und begründet diese Entscheidung.
In Kapitel 7 wird die Grundidee der Umgebung zum Testen der Interaktionskonzepte beschrieben. Zudem ist dort auch eine Beschreibung des Konkreten Spiels zu finden.
Das Grobkonzept in Kapitel 8 zeigt den groben Funktionsumfang der Software in Form von UseCase- und Aktivitätsdiagrammen.
Das darauf folgende Feinkonzept geht genauer auf die Applikationsstruktur ein und zeigt dies anhand von Paket- und Klassendiagrammen.
In Kapitel 10 werden einige interessante Problemlösungen beschrieben, wie die Objektselektion und die Kollisionserkennung in der Testumgebung.

Der Implementierungsabschnitt beschreibt einige in vorangegangenen Kapiteln erwähnte Konzepte genauer, wie beispielsweise die Realisierung der Spielaktionen, sowie das Bewegen der Spielobjekte.
Das Kapitel 12 beinhaltet die Evaluierung der Interaktionskonzepte. Dort wird der Ablauf des Usertests geschildert, sowie der Fragebogen erklärt. Auch die Auswertung der durch den Usertest erhaltenen Daten ist dort zu finden.
Darauf folg die Evaluierung der kompletten Arbeit.
Abschließend werden die wesentlichen Ergebnisse der Arbeit in Kapitel 14 zusammengefasst.

2 Problemanalyse

Zur Steuerung der Interaktionen stehen dem Benutzer auf Android Smartphones folgende Eingabemöglichkeiten zur Verfügung.
Das Display, das zwei Steuerungsachsen bereitstellt, welche durch die Multitouch-Fähigkeit mehrfach verwendet werden können. Zusätzlich steht im Fall des Android Smartphones der Beschleunigungssensor zur Verfügung, welcher im üblichen Gebrauch zwei Bewegungsachsen zur Verfügung stellt.
Insgesamt stehen also sechs Bewegungsachsen zur Verfügung (zwei pro Finger plus zwei durch den Beschleunigungssensor), die benutzt werden können, um sie auf die 3D Umgebung anzuwenden. Es werden hier nur zwei Finger in Betracht gezogen, da nicht alle Android Smartphones mehr als zwei Berührungen auf dem Display verarbeiten können (beispielsweise das Samsung Galaxy 551).
Die Frage ist nun, welche Kombination der Möglichkeiten am natürlichsten wirkt, um Bewegungen durchzuführen.

Allerdings beschreibt ein Interaktionskonzept nicht nur die Bewegung in einer Welt, sondern auch die Interaktion mit der Welt selbst, zum Beispiel gehört das Interagieren mit Objekten in der Welt dazu.
Interagieren ist ein sehr allgemeines Wort, das nicht genau beschreibt, was mit dem Objekt tatsächlich gemacht werden kann. Die Arbeit beschränkt sich darauf Objekte bewegen und drehen zu können. Durch Kollisionen mit anderen Objekten sollen Aktionen hervorgerufen werden können. Diese Interaktion ist gegenüber der realen Welt sehr abstrakt, sollte jedoch im Kontext eines Spiels für diese Arbeit realistisch genug wirken.

Für die Ist-Analyse stützt sich die Arbeit auf eine Kategorisierung von Gesten aus der Literatur **[Bre10]** .

Einige Gesten zur Interaktion in 3D Umgebungen sind in mehreren Interaktionskonzepten zu finden, die ihren Ursprung in der 2D Interaktion finden, diese werden hier kurz beschrieben.

Geste	**Beschreibung der Geste**
Verschiebe-Geste	Mit einem Finger wird ein selektiertes Objekt von einem Ort an seine neue Position verschoben.
Rotations-Geste	Hierbei handelt es sich um eine Zwei-Finger Geste. Das Objekt wird mittels der beiden Finger um die Z-Achse gedreht, wobei der Rotationswinkel zwischen der vorherigen Position und der neuen

	gemessen wird.
Skalierungs-Geste	Bei dieser Geste werden zwei Finger auf das Objekt gelegt und der Abstand zwischen den beiden Fingern, wird entweder verkleinert bzw. vergrößert. Die Größe des Objekts wird entsprechend verkleinert oder vergrößert.
Tap-Geste	Die Tap-Geste dient der Selektion eines Objekts und entspricht dem Klicken der Maus. Hierbei wird das Objekt einfach durch einen Finger auf der Multitouch Oberfläche berührt.
Scroll-Geste	Mittels zwei Fingern wird hier die Kamera in X oder Y Richtung bewegt.
Zoom-Geste	Hier wird die gleiche Geste wie bei der Skalierungsgeste durchgeführt, nur dass diesmal die Finger kein Objekt berühren. Es wird ein Zoom In/Out der Kamera durchgeführt.

Aus einigen diesen Gesten werden komplette Interaktionskonzepte entworfen. Zum Beispiel wird in manchen Konzepten die Zoom-Geste verwendet, um die virtuelle Kamera nach vorne und zurück zu bewegen (z.B. in **4.3 Interaktionskonzept der Kreativagentur NMY**).
Während der Recherche zur Thesis wurde kein Interaktionskonzept gefunden, das diese 2D-Gesten zur vollständigen Bewegung von Objekten benutzt.

3 Grundlagen

Im Grundlagenkapitel werden die Technologien und theoretische Grundlagen vorgestellt, die für die Arbeit von Relevanz sind. Dies dient dem Leser ein besseres Verständnis für die Probleme und Vorgehensweisen dieser Arbeite zu erhalten.

3.1 Android

Da es sich bei dieser Arbeit um die Untersuchung der Interaktionskonzepte auf einem Android Smartphone handelt, werden hier einige Grundlagen zu Android erklärt.

Dieses Unterkapitel baut auf den Informationen aus **[Goo11]** , **[cne11]** und **[ope11]** auf.

Das Android Betriebssystem ist ein Betriebssystem für Handys, Smartphones und Tablet PCs, welches von der Open Handset Alliance entwickelt wurde. Angestoßen wurde die Entwicklung vom Softwarehersteller Google.
Zu den Mitgliedern der Allianz gehören allerdings nicht nur Softwarehersteller, sondern auch Hardware Hersteller und Telekommunikations-Anbieter.

Das Betriebssystem und alle darin enthaltenen Bibliotheken sind quelloffen und können nach Belieben angepasst werden. Einige Smartphonehersteller passen beispielsweise die komplette Benutzeroberfläche auf ihre Wünsche an.

3.1.1 Applikationsstruktur

Die Informationen aus diesem Unterkapitel stammen aus **[Goo111]** .
Applikationen auf Android bestehen typischer Weise aus mindestens einer Activity. Eine Activity ist eine "Seite", auf der verschiedenste Objekte zur Userinteraktion positioniert werden können und deren Logik implementiert wird (beispielsweise Buttons oder Eingabefelder).

Da bei Smartphones auf eine effiziente Ressourcenverwaltung geachtet werden muss, besitzt die Activity einen festgelegten Lebenszyklus, um den Speicher möglichst wenig zu belasten.

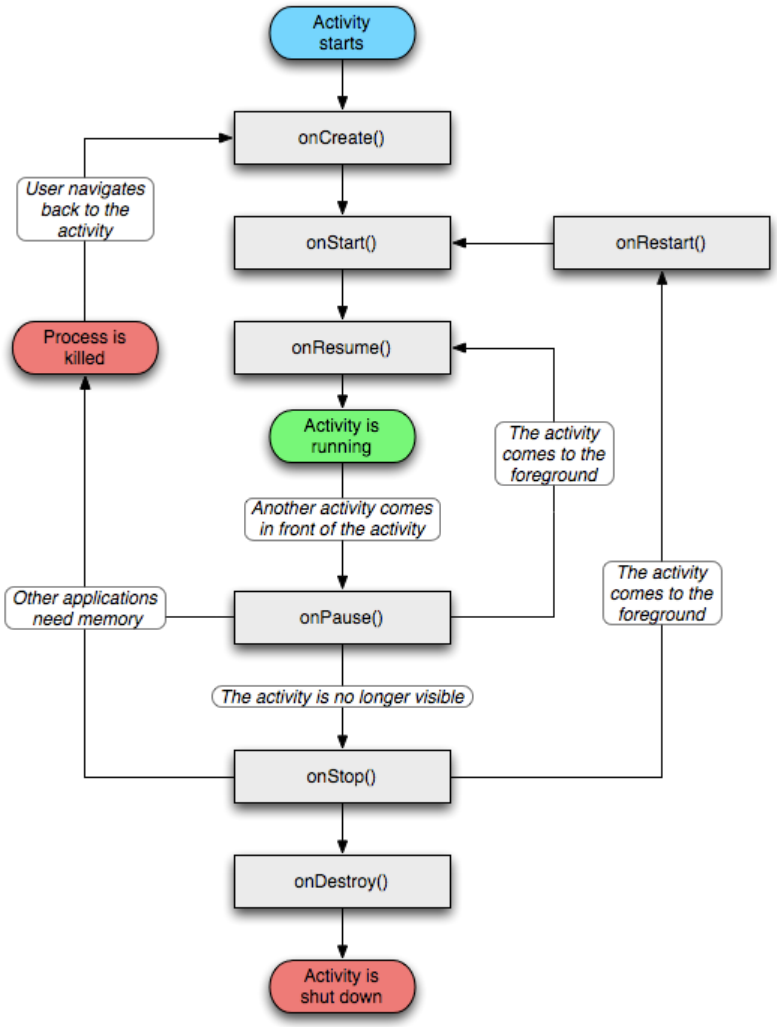

*Abbildung 1: Lebenszyklus einer Activity aus **[Goo111]***

In **Abbildung 1** ist der eben erwähnte Lebenszyklus zu betrachten. Wird eine Activity gestartet, werden die Methoden onCreate(), onStart() und onResume() ausgeführt.
Wird eine Weitere Activity gestartet, wird die vorherige Activity pausiert indem onPause() aufgerufen wird. Ist die erste Activity sogar komplett von der zweiten verdeckt (also nicht mehr sichtbar), wird zusätzlich onStop() aufgerufen.
Werden in dieser Phase Ressourcen knapp, kann die Activity sogar komplett zerstört werden, um wieder Ressourcen freizugeben. Wird dann aber wieder zur zerstörten Activity navigiert, wird diese neu gestartet.
Haben die Ressourcen ausgereicht auch ohne die Activity zu beenden, können bei einer Rückkehr die Methoden onRestart() bzw. onResume() aufgerufen werden und die Activity erscheint, wie sie der User verlassen hat.

Dieser Lebenszyklus muss auch in dieser Arbeit beachtet werden, da beispielsweise die Texturen in der 3D-Umgebung aus dem Speicher gelöscht werden, sobald die zuständige

Activity pausiert wird. Bei der Rückkehr müssen die Texturen in der Methode onResume() erneut geladen werden.

3.1.2 Entwicklung

Software für Android wird zwar in Java entwickelt, allerdings sind nicht alle Libraries vorhanden, die den Entwicklern an PCs bereitstehen. Die GUI-Libraries, wie beispielsweise Swing, fehlen bei Android komplett, da die Oberflächen in Android einen anderen Aufbau haben.

Die Open Handset Alliance stellt den Programmierern ein Software Development Kit (SDK) zur Verfügung, welches kostenlos für Windows, MacOS und Linux herunter geladen werden kann.
Dazu gehören eine API Dokumentation, Tools zum Übersetzen der Programme, sowie ein Emulator zum Testen der Programme.
Im Emulator sind verschiedenen Auflösungen einstellbar, um die Applikation für verschiedenste Bildschirmgrößen testen zu können.
Zudem ist es mittels Telnet möglich dem Emulator Signale zu schicken, welche z.B. einen eingehenden Anruf, oder eine SMS simulieren.
Für Eclipse steht ein Plugin namens „Android Development Tools (ADT) Plugin“ bereit, welches auch einen GUI-Editor beinhaltet, welcher allerdings noch nicht ganz ausgereift ist.
Es ist auch möglich mit Hilfe dieses Plugins Applikationen am Emulator oder am echten Gerät direkt zu debuggen.

Für dieses Unterkapitel wurden die Quellen **[Goo113]** und **[Goo114]** verwendet.

3.2 OpenGL ES

Für das Android Betriebssystem wird die Grafik Bibliothek OpenGL ES zur Verfügung gestellt, um 3D Umgebungen realisieren zu können. Auch in dieser Arbeit wird OpenGL ES verwendet, um das Spiel zum Testen der Interaktionskonzepte zu erstellen.
OpenGL ES steht für Open Graphics Library for Embedded Systems und ist eine Plattformunabhängige Bibliothek zur Entwicklung von 3D Computergrafik. Diese Informationen stammen aus **[KHR11]** .

Es ist eine kompaktere Version von OpenGL. OpenGL ES unterscheidet sich beispielsweise in folgenden Punkten von OpenGL.

In OpenGL konnten Figuren auf folgende Weise gezeichnet werden:

```
glBegin(GL_TRIANGLES);
      glVertex3f(0.0, 6.0, 0.0);
      glVertex3f(-2.0, 0.0, 2.0);
      glVertex3f(2.0, 0.0, 2.0);
glEnd();
```

Code 1: OpenGL - Beispiel zur Zeichnung eines Dreiecks

In **Code 1** ist zu sehen, wie ein Dreieck in OpenGL gezeichnet werden kann. Dazu wird mit glBegin(GL_TRIANGLES) eingeleitet, dass ein Dreieck gezeichnet wird. Danach folgen drei Punkte, die die Eckpunkte des Dreiecks darstellen. Zum Schluss wird glEnd() verwendet, um OpenGL mitzuteilen, dass das Dreieck nun fertig ist.

In OpenGL ES ist dies nun nicht mehr möglich, stattdessen können Figuren wie folgt gezeichnet werden:

```
gl.glVertexPointer(3, GL10.GL_FLOAT, 0, verticesBuffer);
gl.glDrawElements(GL10.GL_TRIANGLES, numOfIndices,
                  GL10.GL_UNSIGNED_SHORT, indicesBuffer);
```

Code 2: OpenGL ES - Beispiel zur Zeichnung einer Figur

In der ersten Zeile in **Code 2** wird OpenGL ES der FloatBuffer verticesBuffer übergeben, welcher in der zweiten Zeile im Dreieckmodus gezeichnet wird. Die Reihenfolge, in der die Punkte zu Dreiecken zusammengesetzt werden, ist im ShortBuffer indicesBuffer festgelegt.

Ein weiterer Unterschied ist, dass OpenGL einen Mechanismus zur Objektselektion liefert, welcher in OpenGL ES nicht vorhanden ist. Da in dieser Arbeit die Objektselektion eine Rolle spielt, muss dafür eine eigene Lösung entwickelt werden, diese wird in **12.7.1 Objektselektion** beschrieben.

Informationen zu den Unterschieden wurden aus **[DGL]** entnommen.

3.3 Multitouch Displays

Eine Eingabemöglichkeit, die dem Benutzer auf dem Android Smartphone zur Verfügung steht, ist das Multitouch-Display, welches auch im Falle dieser Arbeit eine Rolle spielt. Daher wird in diesem Abschnitt anhand eines Zitats erläutert, was ein Multitouch-Display ist und wie es funktioniert.

Folgendes Zitat definiert Multitouch-Displays:

"Ein Touchscreen ist ein Computereingabegerät, meist als kombiniertes Ein-Ausgabegerät, bei dem durch Berührung von Teilen eines Bildes der Programmablauf eines technischen Gerätes, meist eines Computers, direkt gesteuert werden kann. Die technische Umsetzung der Befehlseingabe ist für den Nutzer gleichsam unsichtbar und erzeugt so den Eindruck einer unmittelbaren Steuerung eines Computers per Fingerzeig. Das Bild, welches durch das darauf oder darunter befindliche Touchpad berührungsempfindlich gemacht wird, kann auf verschiedene Weise erzeugt werden: dynamisch mittels Monitoren, über Projektion oder physikalisch (etwa als Ausdruck).

Statt einen Cursor per Maus oder Ähnlichem zu steuern, kann der Finger oder ein Zeigestift verwendet werden. Die Anzeige eines Cursors wird damit überflüssig.

Die Analogie zum Mausklick ist ein kurzes Tippen. Durch Ziehen des Fingers oder Stiftes über den Touchscreen kann eine Drag-and-Drop-Operation ausgeführt werden. Manche Systeme können mehrere gleichzeitige Berührungen zu Befehlen verarbeiten (Multitouch), um zum Beispiel angezeigte Elemente zu drehen oder zu skalieren." **[Wik1]**

3.3.1 Kapazitive Touchscreens

Folgendes Zitat erklärt kapazitive Touchscreens:

"Kapazitive Touchscreens (wie beim iPhone) ermitteln die Position der Berührung durch Veränderung eines elektrischen Feldes. Dazu muss mindestens ein Finger den Touchscreen berühren – mit Handschuhen oder per Stift klappt es nicht. Elektroden in den Ecken erzeugen ein schwaches elektrisches Feld. Bei einer Berührung wird ein Teil der Ladung des Feldes abgeleitet. Dadurch verändert sich das elektrische Feld. Diese Veränderung lässt sich messen – und so die Position des Fingers bestimmen." **[Com]**

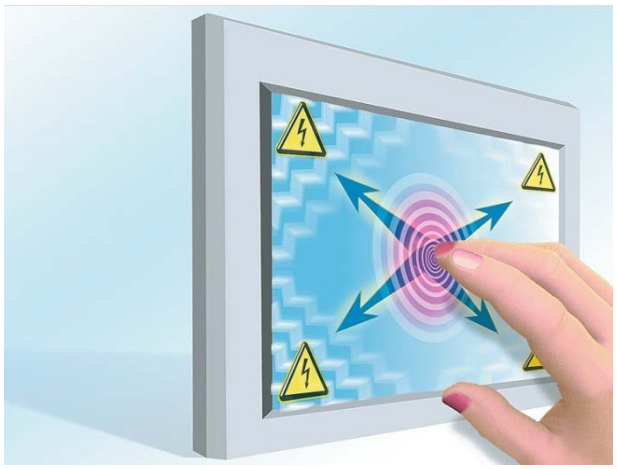

*Abbildung 2: Schaubild eines kapazitiven Touchscreens aus **[Com]***

3.3.2 Resistive Touchscreens

Folgendes Zitat erklärt resistive Touchscreens:

"Resistive Touchscreens erkennen die Berührung durch leichten Druck auf eine Spezialfolie. Auf der Bildschirmoberfläche liegt eine elektrisch leitende Folie. Darüber liegt eine zweite Folie, durch mikroskopisch kleine Abstandshalter von der ersten Folie getrennt. Wird die zweite Folie durch eine Berührung des Touchscreens heruntergedrückt, fließt ein schwacher Strom. Das funktioniert per Hand und Stift. Die Herstellung ist relativ günstig, deshalb stecken sie zum Beispiel in vielen preiswerten Navigationsgeräten. Sie sind empfindlicher gegenüber Kratzern." **[Com]**

*Abbildung 3: Schaubild eines resistiven Touchscreens aus **[Com]***

3.4 Beschleunigungssensor

Eine weitere Eingabemöglichkeit, die dem Benutzer auf dem Android Smartphone zur Verfügung steht, ist der Beschleunigungssensor, der feststellen kann in welcher Lage sich das Smartphone befindet. Er spielt auch im Falle dieser Arbeit eine Rolle, daher wird er in diesem Abschnitt anhand eines Zitats erläutert.

Folgendes Zitat definiert den Beschleunigungssensor:

"Ein Beschleunigungssensor ist ein Sensor (Fühler), der die Beschleunigung misst, indem die auf eine Testmasse wirkende Trägheitskraft bestimmt wird. Somit kann z. B. bestimmt werden, ob eine Geschwindigkeitszunahme oder -abnahme stattfindet. [...]

Er wird auch Beschleunigungsmesser oder Accelerometer genannt, weiterhin B-Messer und G-Sensor. Werden kontinuierliche Beschleunigungsmessungen aufgezeichnet, so bezeichnet man diese Messreihe als Akzelerogramm (Analog zum Seismogramm, das durch ein Seismometer aufgezeichnet wird).

In den letzten Jahren haben miniaturisierte Beschleunigungssensoren zunehmend Bedeutung erlangt. Diese sind mikro-elektro-mechanische Systeme (MEMS) und werden meist aus Silicium hergestellt. Diese Sensoren sind Feder-Masse-Systeme, bei denen die „Federn“ nur wenige µm breite Silicium-Stege sind und auch die Masse aus Silizium hergestellt ist. Durch die Auslenkung bei Beschleunigung kann zwischen dem gefedert aufgehängten Teil und einer festen Bezugselektrode eine Änderung der elektrischen Kapazität gemessen werden. Der gesamte Messbereich entspricht einer Kapazitätsänderung von nur ca. 1 pF, daher muss die Elektronik zur Auswertung dieser kleinen Kapazitätsänderung gleich auf demselben Halbleiterbaustein integriert werden." **[Wik]**

3.5 Usability

Ein wesentlicher Bestandteil dieser Thesis ist es herauszufinden welches der Konzepte den Benutzern eher zusagt, das heißt welches benutzerfreundlicher ist.
Um die Benutzerfreundlichkeit (oder auch Usability genannt) einer Software zu bewerten, ist eine Usertest unumgänglich.
Auch im Rahmen dieser Thesis wird ein Usertest durchgeführt, um die Benutzerfreundlichkeit der beiden Interaktionskonzepte zu evaluieren.

Alle in diesem Kapitel (samt Unterkapiteln) befindlichen Informationen stammen aus **[Dör09]** , sofern nicht anders angegeben.

Im Folgenden werden Methoden vorgestellt die Usability zu testen.

3.5.1 Thinking Aloud Test

Beim Thinking Aloud Test testet der User die Software und muss dabei einem Kommentator immer sagen, was er gerade denkt. Der Kommentator ist dafür zuständig das Gesagte des Testers zu protokollieren und den Testuser daran zu erinnern, seine Gedanken zu verbalisieren.
Da das ständige verbalisieren für viele Menschen ungewohnt ist, ist die Erinnerung des Kommentators nötig. Ansonsten darf der Kommentator nicht in das Geschehen eingreifen.

3.5.2 Fragebögen

Eine weitere Möglichkeit ist es dem Benutzer, nachdem er die Software getestet hat, einen Fragebogen zu geben. Die Fragebögen werden später ausgewertet und können Aussagen über die Software bestätigen bzw. für falsch erklären. Dazu später mehr, zuerst aber die verschiedenen Möglichkeiten Fragebögen aufzusetzen.

3.5.2.1 Offene Fragen

Zum einen können offene Fragen gestellt werden, welche allerdings nicht gut in eine Statistik aufgenommen werden können. Es ist lediglich möglich festzustellen, ob die Benutzer ähnliche Aussagen über die Software machen.

Eine beispielhafte Frage wäre: "Welche Verbesserungsvorschläge haben Sie?"

3.5.2.2 Likert scale

Bei Likert scale werden Behauptungen aufgestellt, die mit einer von fünf vorgegebenen Antworten bewertet werden können.

Ein Beispiel:

(1) Stimmt gar nicht
(2) Stimmt meist nicht
(3) Weiß nicht
(4) Stimmt meist

(5) Stimmt genau
Die Menüs ließen sich leicht selektieren: __
Entfernungen konnte man nicht abschätzen: __

3.5.2.3 Semantic differential scale

Bei dieser Methode, werden zu einer Frage oder Behauptung zwei Gegensätze aufgezeigt. Zwischen den Gegensätzen kann bei einem Intervall von 1-7 die Tendenz angegeben werden.

Ein Beispiel:

Die Bedienung der Software ist:
verständlich _ _ _ _ _ _ _ unverständlich

3.5.2.4 Anordnungsfragen

Bei Anordnungsfragen werden zu einer Frage Antworten aufgezeigt, die nach Priorität bewertet werden müssen.

Ein Beispiel:

Wie häufig nutzen die folgenden Möglichkeiten, Hilfe zu erhalten?
(1 – am häufigsten; 4 – am wenigsten häufig)

Button auf Flystick __
Kontextmenü durch Geste __
Sprachkommando „Hilfe“ __
Ausgedruckte Help-Karte __

3.5.2.5 Multiple-Choice Fragen

Dem Benutzer werden zu einer Frage mehrere Antwortmöglichkeiten angezeigt, wobei er alle zutreffenden Antworten ankreuzen muss.

Ein Beispiel:

Welche Funktionen der Steuerung haben Sie genutzt? (Kreuzen Sie bitte alles Zutreffende an)

Flystick □
Spacemouse □
Konventionelle Maus □
Keyboard □
Spracheingabe □
Data Glove □

3.5.3 Auswertung

Offene Fragen liefern Anectodal Evidence.

"Unter dem Begriff »Anecdotal Evidence« versteht man Aussagen, Belege, Begründungen und Beweise, die nicht statistisch oder auf andere Weise wissenschaftlich nachprüfbar sind, also lediglich die subjektiven Erlebnisse bzw. Schilderungen einzelner Personen widerspiegeln. [...]" **[onp11]**

Bei Skalaren können statistische Größen ermittelt werden, wie der Erwartungswert (Was wurde im Durchschnitt geantwortet?) oder die Standardabweichung (Waren sich die Testuser einig, oder ging deren Meinung stark auseinander?).

3.5.3.1 Visualisierung der Ergebnisse

Die Ergebnisse können beispielsweise als Tukey-Box-Plot visualisiert werden. Ein Box-Plot (oder auch Kastengrafik) wird verwendet, um die Verteilung statistischer Daten darzustellen.

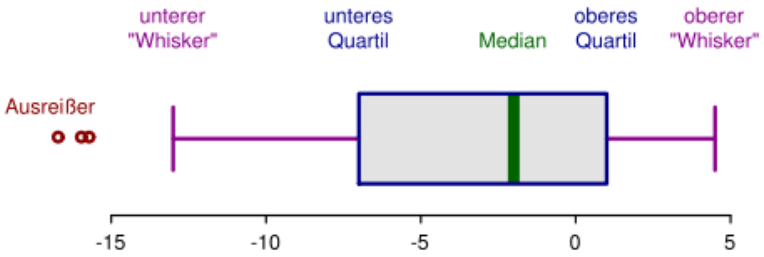

*Abbildung 4: Beispiel eines Box-Plots. Leicht bearbeitet aus **[Wik2]***

In **Abbildung 4** ist ein Box-Plot samt Bezeichnungen seiner Grenzen zu sehen. Das graue Rechteck in der "Mitte" wird als Box bezeichnet und startet mit dem so genannten unteren Quartil und endet mit dem oberen Quartil.
Zwischen dem oberen und unteren Quartil befinden sich die mittleren 50% der Daten. Diese 50% werden wiederum durch den Median in der prozentualen Mitte getrennt. Das heißt links des Median liegen die kleineren 25% und rechts daneben die größeren 25%.

Durch die Whisker werden die außerhalb der Box liegenden Werte dargestellt. Die Whisker haben die maximallänge der 1,5 fachen Box-Länge. Befindet sich kein Wert direkt auf der 1,5 fachen Länge der Box wird der nächst nähere Wert an der Box als Whisker gesetzt.

Alle anderen Werte, die außerhalb der Whisker liegen, werden als Ausreißer bezeichnet.

Der Inhalt dieses Unterkapitels basiert auf **[Wik2]** .

3.5.3.2 Durchführung statistischer Tests

Lässt man alle Benutzer (mit alle Benutzer sind alle Benutzer gemeint, die die Software später benutzen werden) den Benutzertest durchführen, werden keine statistischen Tests benötigt, um die Korrektheit der dadurch entstandenen Aussagen belegen zu können. Statistische Tests werden nur benötigt, wenn von einer Stichprobe auf die Gesamtheit geschlossen werden soll.
Bei einer Stichprobe kann es passieren, dass diese nicht repräsentativ für die Gesamtheit ist. Deshalb berechnen die statistischen Tests die Irrtumswahrscheinlichkeit, um herauszufinden, ob die Aussagen für die Gesamtheit gültig sind. In der Regel gelten 5% Irrtumswahrscheinlichkeit als akzeptabel. Ist der Wert größer, kann keine Aussage getroffen werden.

Im Folgenden wird erläutert, welche Art von statistischen Tests für die Auswertung der Daten, die mit Hilfe des während der Thesis durchgeführten Usertests gewonnen wurden, in Frage kommen.
Der Fragebogen, der nach dem Usertest ausgehändigt wurde, verwendet hauptsächlich die Semantic differential scale Methode.
Da es sich bei dem Usertest um zwei verbundene Stichproben handelt (alle User testen beide Interaktionskonzepte), kann der Vorzeichentest sowie der Wilcoxon-Mann-Whitney Test in Betracht gezogen werden.
Der Vorzeichentest ist sehr Robust, aber nicht effizient - die Größe der Unterschiede werden nicht berücksichtigt.
Die für den Fall dieser Thesis gewählte Alternative ist der Wilcoxon-Mann-Whitney Test, welcher den Betrag der Differenz der Werte berücksichtigt. Ein weiterer Vorteil ist, dass dieser Test für verbundene und nicht verbundene Stichproben verwendet werden kann, was allerdings in diesem Fall keine Rolle spielt.

Zur statistischen Analyse kann Software verwendet werden, wie beispielsweise Excel, *R* oder SPSS.

4 Beispielhafte Untersuchung vorhandener Interaktionskonzepte

Teil dieser Arbeit ist es vorhandene Interaktionskonzepte zu untersuchen, um die Ist-Situation besser einschätzen zu können und um Teile der vorhandenen Konzepte für die Verwendung eigener Konzepte vorzumerken. Dies geschieht in diesem Kapitel.

4.1 The Z technique

Die "Z technique" (übersetzt: z-Technik), welche in folgender Quelle gefunden wurde **[Ant10]** , ermöglicht dem Benutzer das Bewegen eines Objekts in einem dreidimensionalen Raum.

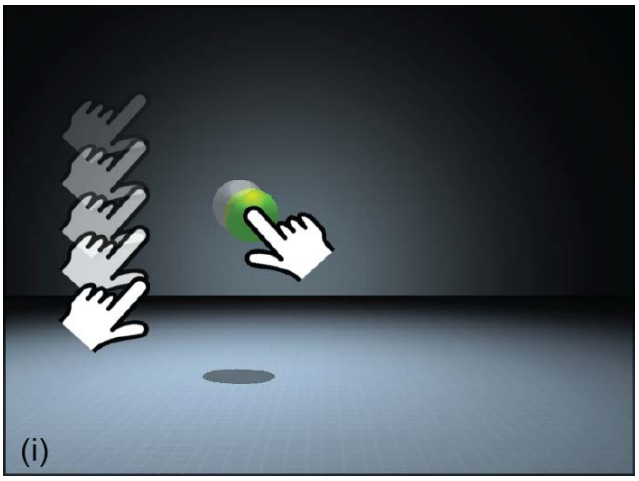

*Abbildung 5: The Z technique aus **[Ant10]***

Das zu bewegende Objekt wird hierbei mit dem Finger ausgewählt und kann parallel zur Projektionsebene in zwei Dimensionen verschoben werden. Mit einem zweiten Finger ist es möglich das ausgewählte Objekt durch hoch- und runter-Bewegungen in der Tiefe zu verschieben (angedeutet in **Abbildung 5**). Allerdings gibt es in diesem Konzept keine Möglichkeit die Kamera zu bewegen, oder auf sonstige Weise mit dem Objekte zu interagieren.

Die z-Technik schafft es Objekte mit relativ simplen Gesten in drei Dimensionen zu bewegen. Das Bewegen der Objekte parallel zur Projektionsebene könnte man mit dem Bewegen eines realen Objekts auf einer Platte assoziieren.
Denkbar ist es dieses Konzept mit einem anderen Bewegungskonzept zu kombinieren, um eine Bewegung der Kamera zusätzlich möglich zu machen.
Durch Hinzufügen weiterer Gesten könnte das Objekt auch weiterführend manipuliert werden, beispielsweise könnten Rotationen ermöglicht werden.

4.2 Multitouch viewports

"Multitouch viewports", abgeleitet von der "four views technique", zeigt eine Szene aus vier verschiedenen Perspektiven. Im Normalfall sind alle vier Ansichten gleich groß und benutzen dasselbe Sichtfeld für die virtuellen Kameras. In jeder Perspektive kann ein Objekt in zwei Dimensionen parallel zur Projektionsebene bewegt werden.
"Multitouch viewports" unterscheidet sich von der "four views technique" darin, dass man aus mehreren Ansichten gleichzeitig agieren kann (wie in **Abbildung 6** mit den eingezeichneten Händen angedeutet). Auch in diesem Konzept gibt es keine Möglichkeit die Kamera zu bewegen. Gefunden wurde dieses Konzept in **[Ant10]** .

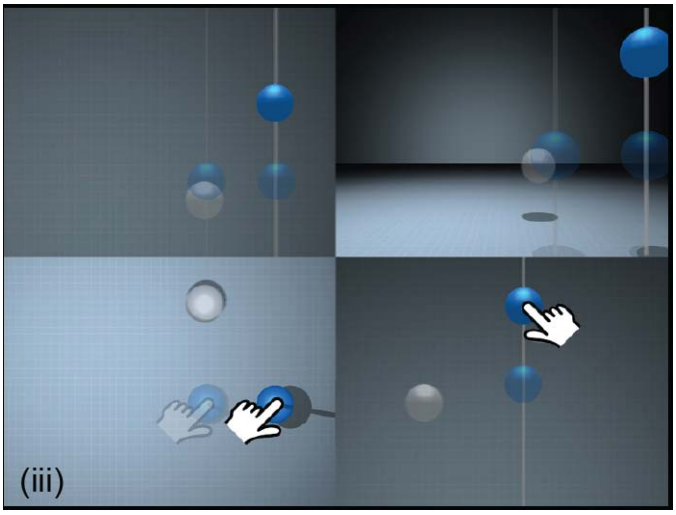

*Abbildung 6: Multitouch viewports aus **[Ant10]***

Zur Verdeutlichung der Technik ist in **Abbildung 6** ein Screenshot zu sehen. Folgende Perspektiven sind zu erkennen:

Oben links:	Kamera blickt in Richtung -z.
	Das Objekt kann parallel zur xy-Ebene bewegt werden.
Oben rechts:	Kamera blickt in eine beliebige Richtung, in diesem Fall ebenfalls -z.
	Das Objekt kann parallel zum Boden (xz-Ebene) bewegt werden.
Unten links:	Kamera blickt in Richtung -y.
	Das Objekt kann parallel zur xz-Ebene bewegt werden.
Unten rechts:	Kamera blickt in Richtung x.
	Das Objekt kann parallel zur yz-Ebene bewegt werden.

Mit "multitouch viewports" ist es möglich Objekte im dreidimensionalen Raum zu bewegen. Allerdings sind in diesem Konzept ebenfalls keine weiteren Interaktionen beschrieben.
Zudem wird für die vier verschiedenen Ansichten viel Platz benötigt, um diese anzuzeigen. Dieser Umstand mach die Technik für Smartphones kompliziert, da diese im Allgemeinen vergleichsweise kleine Displays haben.
Möglicherweise ist dieses Konzept auch nicht optimal, um eine *Immersion* herbeizuführen, da zwei Ansichten auf eine Szene in der Natur nicht vorkommen und daher eher unnatürlich erscheinen könnten.

Denkbar wäre eine Erweiterung des Konzepts, sodass Rotationen und Skalierungen durch Multitouch-Gesten ermöglicht werden.
Für kleine Displays könnten die vier Views auf zwei reduziert werden, auf diese Weise wäre dennoch eine 3D Transformation möglich.

Trotzdem würde aber das Problem bestehen die Kamera zu bewegen. Dieses Konzept ist für eine weiterführende Verwendung eher uninteressant.

4.3 Interaktionskonzept der Kreativagentur NMY

"NMY realisierte im Jahr 2009 ein 3D Multitouch Präsentationssystem für den Kunden EADS. Es war bis dato das erfolgreichste Messeexponat in der 10 jährigen EADS Historie. Für den weltweiten Einsatz auf Messen und Showrooms sollte nun die Applikation in eine noch umfassendere und realistischere Ebene geführt werden. Für das Jahr 2010 erhält NMY den Auftrag das Multitouch 3D Präsentationssystem mit Produkten, Systemen, Daten, Statistiken, politischen und Marketingbotschaften in die nächste Ebene zu führen. Die 3D Welt der EADS soll sukzessive für Marketing und Vertrieb auf Messen, Showrooms und 1:1 Präsentationen auf großen wie kleinen Multitouch Devices präsentiert werden können." ***[Kre11]***

Mit dem Interaktionskonzept der Kreativagentur NMY (aus **[Kre11]**) ist es Möglich die virtuelle Kamera in einem dreidimensionalen Raum frei zu bewegen. Die virtuelle Kamera kann mit Gesten geneigt, gedreht und dreidimensional transformiert werden.
Die visuelle Kennzeichnung mancher Objekte (zu sehen in **Abbildung 7**) deutet darauf hin, dass man mit ihnen interagieren kann, bestätigt wird dies allerdings durch das Video nicht.

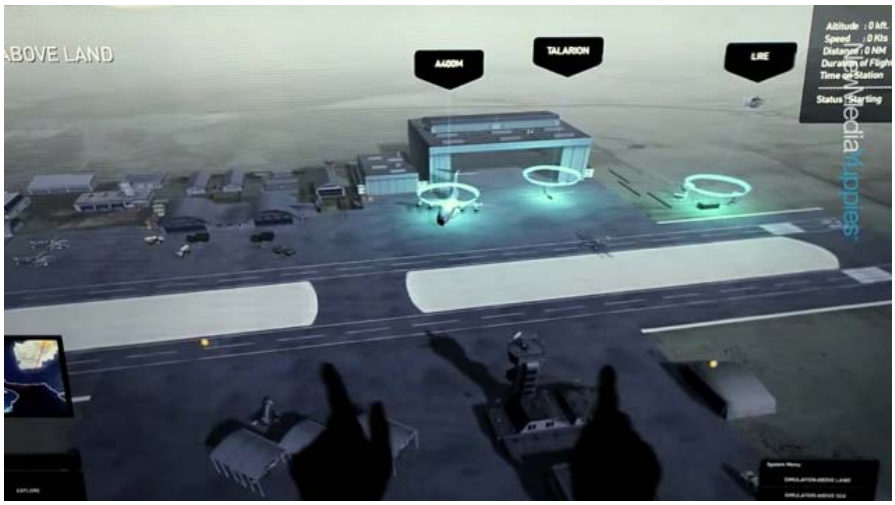

*Abbildung 7: 3D Multitouch Präsentationssystem von **[Kre11]***

NMY hat keine schriftliche Beschreibung ihres Konzepts, allerdings stellen sie ein Video zur Verfügung, aus dem man folgende Interaktionsmöglichkeiten ablesen kann:

Interaktion	**Beschreibung der Geste**
Kamera vorwärts/rückwärts bewegen	Es werden zwei Finger auf dem Screen auseinander geschoben, um eine Bewegung nach vorne zu ermöglichen, bzw. werden die Finger zusammen geschoben, um eine Bewegung zurück zu ermöglichen.
Kamera drehen	Um die virtuelle Kamera zu drehen, wird ein Finger auf dem Screen positioniert und der zweite Finger führt mit einer Kreisbewegung um den stehenden Finger eine Drehung durch.
Kamera neigen	Die virtuelle Kamera wird geneigt, indem beide Finger horizontal nebeneinander nach oben, oder unten bewegt werden.
Kamera seitliche bewegen	Für eine Seitliche Bewegung werden beide Finger horizontal nebeneinander angeordnet nach rechts bzw. links bewegt.

Das 3D Multitouch Präsentationssystem von NMY ist ein Beispiel zur Bewegung der virtuellen Kamera im dreidimensionalen Raum mit Hilfe eines Multitouch Eingabegeräts.
Kombiniert mit der z-Technik (aus Abschnitt **4.1 The Z technique**) könnte dadurch möglicherweise auch gut mit Objekten interagiert werden.
Zu bemängeln ist allerdings, dass durch das Auseinanderziehen der Finger die Bewegung recht ungenau ist.
Insgesamt scheint dieses Konzept punktgenaue Bewegungen nicht zuzulassen, daher wird es für die Grundlage für ein eigenes Interaktionskonzept nicht in Betracht gezogen.

4.4 Interaktionskonzepte diverser Android Spiele

In einigen Android Spielen, wie beispielsweise NOVA, Modern Combat Sandstorm oder Dungeon Hunter, werden zweidimensionale Steuerelemente eingeblendet, die zur Interaktion mit der Welt dienen. Die Steuerelemente sind beispielsweise virtuelle Steuerknüppel, die für eine zweidimensionale Bewegung eingesetzt werden. In Modern Combat Sandstorm wird der Steuerknüppel zum Beispiel eingesetzt, um die Spielfigur nach vorne, zurück und seitlich zu bewegen (auch strafen genannt).
Des Weiteren gibt es Buttons, die Aktionen wie das Schießen, Schlagen, oder diverse Interaktionen mit Objekten auslösen. Informationen zu den genannten Spielen stammen aus folgenden Quellen: **[Gam11]** ,**[Gam111]** und**[Gam112]**

*Abbildung 8: Screenshot des Spiels ModernCombat aus **[Gam111]***

Abbildung 8 ist ein Screenshot von Modern Combat Sandstorm. Dort ist links unten der bereits erwähnte Steuerknüppel eingeblendet, mit dem sich die Spielfigur bewegen lässt. Der runde Button rechts unten ist zum schießen. Der Spieler schaut sich um, indem er seinem Finger auf die 3D Welt hält und in die Richtung schiebt, in die er sich drehen möchte.

Prinzipiell benutzt dieses Konzept eine altbewährte Technik, um mit der Welt zu interagieren - es stellt dem Spieler ein Gamepad zur Verfügung, allerdings ein virtuelles.
Dies hat den Vorteil, dass vielen bekannt ist, wie ein Gamepad zu benutzen ist, allerdings hat es auch einen erheblichen Nachteil gegenüber eines echten Gamepads - man spürt die Buttons auf dem Touchscreen nicht.

5 Eigene / abgeleitete Interaktionskonzepte

In diesem Kapitel werden die, durch den Autor, erarbeiteten Konzepte vorgestellt. Die Namen der Konzepte sind frei erfunden.
Alle Interaktionskonzepte sollen dem Benutzer ermöglichen sich in einer dreidimensionalen Umgebung zu bewegen und mit Objekten zu interagieren.
Dabei soll sich die virtuelle Kamera immer in "Kopfhöhe" vom Boden entfernt befinden. Es ist möglich sich parallel zum Boden zu bewegen und sich in alle Richtungen umzuschauen. Zusätzlich soll mit Objekten interagiert werden können. Das schließt ein, dass sie im Raum bewegt werden. Auch Rotationen sollen möglich sein.

5.1 AccMove

Der Name "AccMove" eine Kombination aus Accelerometer und move, da in diesem Konzept der Beschleunigungssensor (deutsch für Accelerometer) eine erhöhte Rolle spielt. Dieser wird verwendet, um Bewegungen der Spielfigur sowie der der Objekte durchzuführen.

Interaktion	Beschreibung der Geste
Kamera vorwärts/rückwärts bewegen	Die Kamera kann nach vorne und zurück bewegt werden, indem das Smartphone nach vorne oder hinten geneigt wird.
Kamera drehen	Smartphone nach rechts oder links neigen. Oder mit dem Finger außerhalb eines Objekts nach links oder rechts wischen.
Kamera neigen	Mit dem Finger außerhalb eines Objekts nach oben oder unten wischen.
Objekt nehmen	Ein Objekt wird genommen, indem das Objekt mit dem Finger berührt wird.
Objekt horizontal und vertikal verschieben	Ein Objekt kann horizontal und vertikal verschoben werden, indem es mit dem Finger selektiert und mit diesem verschoben wird.
Objekt in der Tiefe verschieben	Objekt mit dem Finger berührt halten und das Smartphone nach vorne oder hinten neigen.
Objekt um die y-Achse rotieren	Ein Objekt kann rotiert werden, indem das Smartphone nach links oder rechts geneigt wird, während das Objekt mit dem Finger gehalten wird.

Ein Objekt fallen lassen	Wenn das Objekt mehrmals hin und her geschoben wird, wird es fallen gelassen.

5.2 TabAndGo

Dieses Interaktionskonzept nennt sich "TabAndGo" und leitet sich teilweise von der Z technique ab. Möglicherweise finden sich auch Gesten aus anderen Konzepten wieder, allerdings sind sie nicht abgekupfert, sondern für diese Arbeit entwickelt. Da die Gesten sehr einfach sind, ist es sehr wahrscheinlich, dass sie auch schon in anderen Projekten genutzt werden.

Interaktion	**Beschreibung der Geste**
Kamera bewegen	Durch das doppelte Anklicken einer Stelle auf dem Boden, soll sich die Kamera automatisch dort hin bewegen.
Kamera vorwärts/rückwärts bewegen	Wird ein Finger auf dem Display gehalten, kann mit dem anderen Finger die Kamera vor und zurück gezogen werden, vergleichbar mit der Z technique.
Umschauen (Kamera drehen/neigen)	Um die Kamera zu drehen und zu neigen, kann mit einem Finger in die entsprechende Richtung gewischt werden. Die Kamera dreht sich beispielsweise nach rechts, wenn nach links gewischt wird.
Objekt nehmen	Ein Objekt wird genommen, indem es mit dem Finger berührt wird.
Objekt horizontal und vertikal verschieben	Ein Objekt kann horizontal und vertikal verschoben werden, indem es mit dem Finger selektiert und mit diesem verschoben wird.
Objekt in der Tiefe verschieben	Ein Objekt kann in der Tiefe verschoben werden, indem es mit einem Finger selektiert wird und mit einem anderen Finger eine Schiebe-Geste außerhalb des Objekts durchgeführt wird.
Objekt um die y-Achse rotieren	Ein Objekt kann rotiert werden, indem das Smartphone nach links oder rechts geneigt wird.
Ein Objekt fallen lassen	Durch das Halten eines Fingers auf ein Objekt, erscheint ein Menü, das dem Benutzer ermöglicht das Objekt fallen zu lassen.

Die Hauptmotivation für dieses Interaktionskonzept ist es möglichst ohne zweidimensionales Overlay für Interaktionen auszukommen, ganz im Gegensatz zur Beschreibung aus **4.4 Interaktionskonzepte diverser Android Spiele**.

5.3 vPad

Der Hauptunterschied zu den andern Konzepten soll darin bestehen, dass die Gesten nicht mehr direkt auf der 3D Umgebung stattfinden, sondern eine virtuelle Steuerkonsole eingeblendet wird, mit der alle Interaktionen gesteuert werden.

Die virtuelle Steuerkonsole wird wie folgt aussehen: Unten links auf dem Display wird ein Steuerknüppel dargestellt, welcher dem Benutzer ermöglicht nach vorne und zurück zu gehen. Unten rechts ist ein weiterer Steuerknüppel, welcher dem Benutzer erlaubt sich umzuschauen.
Interagiert der Benutzer gerade mit einem Objekt, dient der linke Steuerknüppel dazu das Objekt in der Tiefe zu verschieben. Der Rechte Steuerknüppel kann das Objekt zweidimensional parallel zur Ansicht bewegen.

Interaktion	**Beschreibung der Geste**
In den Bewegungs-modus wechseln.	(Ist standardmäßig aktiv) Ansonsten: Neben ein Objekt "klicken" oder das Objekt fallen lassen.
In den Objekt-bewegungsmodus wechseln	Ein Objekt berühren.
Nur im Bewegungsmodus möglich	
Kamera vorwärts/rückwärts bewegen	Nach vorne und zurück kann sich bewegt werden, indem der linke Steuerknüppel nach vorne bzw. hinten geschoben wird.
Umschauen (drehen/neigen)	Umschauen ist mit dem rechten Steuerknüppel möglich. Schiebt man ihn nach rechts, schwenkt die virtuelle Kamera nach rechts.
Objekt nehmen	Ein Objekt wird genommen, indem das Objekt mit dem Finger berührt wird.
Nur im Objektbewegungsmodus möglich	
Objekt horizontal und vertikal verschieben	Möglich mit dem rechten Steuerknüppel.
Objekt in der Tiefe verschieben	Möglich mit dem linken Steuerknüppel.

Objekt um die y-Achse rotieren	Ein Objekt kann rotiert werden, indem das Smartphone nach links oder rechts geneigt wird.
Ein Objekt fallen lassen	Das Objekt wird abgewählt, wenn es erneut mit dem Finger berührt wird.

6 Auswahl der Konzepte zur Implementierung

Ein Bestandteil der Thesis ist es zwei Interaktionskonzepte gegenüberzustellen, um herauszufinden, welches eine bessere Userakzeptanz erzielt.
Die Gegenüberstellung wurde auf zwei Konzepte begrenzt, da es ansonsten den Rahmen der Thesis sprengen würde. Die Vorauswahl wird vom Verfasser der Thesis getroffen.

Während der Recherche wurde deutlich, dass einige Konzepte lediglich ein altbekanntes Konzept simulieren. In einigen 3D-Spielen, welche für Android vorhanden sind, steht dem Benutzer ein virtuelles Gamepad zur Verfügung, um zu interagieren.
Denkbar ist es, dass sich damit die Spiele nicht gut steuern lassen, da man die Schaltflächen nicht spürt und daher nicht genau bedienen kann.

Daher wurden die Interaktionskonzepte "TabAndGo" und "vPad" ausgewählt, um sie im Rahmen eines Usertests zu vergleichen.
"TabAndGo" zeichnet sich dadurch aus, dass es größtenteils auf virtuelle Buttons und Steuerelement verzichtet und es dem Benutzer möglich ist mit logischen Gesten direkt mit der Welt zu interagieren.

Das absolute Gegenteil stellt das Konzept "vPad" dar, welches ausschließlich auf virtuelle Steuerelemente vertraut.

Allerdings stellen beide Konzepte dem Benutzer dieselben Interaktionen zur Verfügung, dies macht die Konzepte möglicherweise besser vergleichbar.

Das Konzept "AccMove" wurde nicht gewählt, da die Verzögerung des Beschleunigungssensors doch spürbar ist und evtl. als störend empfunden wird.

7 Konzeption eines Spiels als Testumgebung

Um die Interaktionskonzepte zu testen, muss zuerst eine Umgebung erstellt werden, in der dies möglich ist. Im Fall dieser Thesis soll ein Spiel erstellt werden, um die beiden ausgewählten Konzepte gegenüberzustellen.

7.1 Grundidee

Prinzipiell bringen beide Interaktionskonzepte dieselben Möglichkeiten mit sich, sodass für beide Konzepte dieselbe Testumgebung / dasselbe Spiel verwendet werden kann.

Dazu wird ein Spiel erstellt, in dem eine oder mehrere Aufgaben gelöst werden sollen. Dieses Spiel wird dann mit beiden Interaktionskonzepten gespielt.
Denkbar wäre auch für beide Konzepte verschiedene Aufgaben zu erstellen, die sich im gleichen Schwierigkeitsbereich aufhalten, um für den Benutzer eine Abwechslung zu schaffen. Zudem würde nur eine einzige Aufgabe die Herausforderung beim Testen des zweiten Konzepts abschwächen, da der Benutzer die Lösung bereits kennt und sich nur noch auf die Steuerung konzentrieren muss.

Aufgrund dieser Vermutung wird es ein Spiel mit zwei Levels geben. Das gewünschte Level kann bei Spielbeginn gewählt werden. Allerdings sind die Interaktionskonzepte an ein Level gebunden. Es ist also nicht möglich ein Level mit beiden Konzepten zu spielen.

Fest steht, dass das Spiel aus der Ego-Perspektive gespielt wird.
Da es zudem möglich ist in beiden Konzepten mit Objekten zu interagieren, sollten Objekte spielerisch eingebaut werden, um Aufgaben zu lösen.

Ein Adventure-Puzzle würde sich gut anbieten, um die Interaktionskonzepte zu testen, da der Benutzer durch die spieltypisch eher ruhige Spielweise nicht unter Druck gesetzt wird.
Auch die Geschicklichkeit und Reaktionszeit des Spielers spielt keine erhöhte Rolle im Gegensatz zu beispielsweise Ego-Shootern. In Ego-Shootern sind zum Sieg oft viel Geschick und Reaktion von Nöten.

Der Spieler startet eingesperrt in einem Bereich (ein Raum, oder ein eingezäuntes Außengelände) und muss versuchen mit Hilfe vorhandener Gegenstände einen Ausweg zu finden.

Objekte sollen aufeinander reagieren können, um das Spiel voranzutreiben.

Ein Beispiel wäre, dass ein Hammer mit vor- und zurück-Bewegungen gegen ein Fenster gehauen werden kann, um es zu zerstören und der Benutzer dadurch an neue Objekte gelangen kann.
Dabei soll auch die Rotation der Objekte eine Rolle spielen: Wird der Hammer falsch herum gehalten (z.B. Stil nach vorne), soll das Einhauen eines Fensters mit ihm nicht möglich sein.
Ein weiteres Bespiel wäre ein Schlüssel, der in ein Schlüsselloch gesteckt werden muss, um das Schloss zu öffnen.

Üblich in ähnlichen Spielen, wie beispielsweise Mystique, ist, dass Objekte in einem Inventar platziert werden und man nicht wirklich mit ihnen interagieren kann. Sie können lediglich ausgewählt werden, um sie mit anderen Gegenständen zu kombinieren. Welche Interaktion zwischen den Objekten ausgeführt wird, entscheidet das Spiel, nicht der Spieler.

7.2 Konkretes Spiel

Das Spiel, das auf den Namen InteractionGame getauft wurde, wird aus zwei Levels bestehen, welche im Folgenden beschrieben sind.

Abbildung 9: Screenshot des ersten Levels

Das erste Level (zu sehen in **Abbildung 9**) setzt das Interaktionskonzept TabAndGo ein. Der Spieler befindet sich in einem Garten hinter einem Haus und hat sich ausgesperrt. Zu sehen ist eine Kiste, auf der sich ein Deckel befindet. Dieser Deckel muss von der Kiste genommen werden, um den darin befindlichen Schlüssel selektieren zu können.
Der Schlüssel muss zum Haus, genauer an die Tür des Hauses, transportiert werden. Zudem muss der Schlüssel in die richtige Lage gedreht werden. Wird der Schlüssel in das Schlüsselloch der Tür gesteckt, öffnet sich diese.
Geht der Spieler durch diese Tür, ist das Level geschafft.

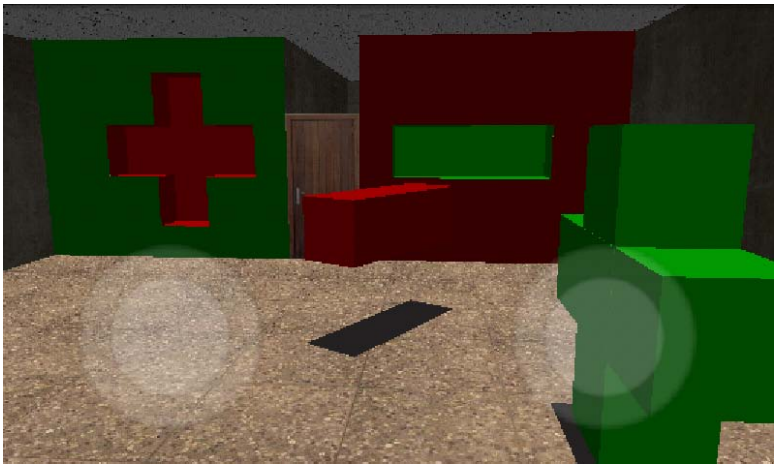

Abbildung 10: Screenshot des zweiten Levels

Das zweite Level (zu sehen in **Abbildung 10**) verwendet vPad als Interaktionskonzept. Der Spieler befindet sich in einem Raum, welcher eine Tür hat. Neben der Tür sind Aussparungen in Form eines Plus- und eines Minuszeichens. Schaut der Spieler sich um, entdeckt er Objekte in Form eines Plus- und Minuszeichens.
Führt der Spieler die Objekte in die Richtigen Aussparungen, öffnet sich die Tür. Geht der Spieler hindurch, hat er das Level gewonnen.

Die Texturen, die im Spiel verwendet werden stammen aus **[tex11]** und **[The]**

8 Grobkonzept

Im Grobkonzept werden die grundlegenden Funktionen der Testumgebung beschrieben.

8.1 Use case-Diagramm

Im Use case-Diagramm werden die grundlegenden Funktionalitäten des Systems auf hohem Abstraktionsniveau beschrieben.

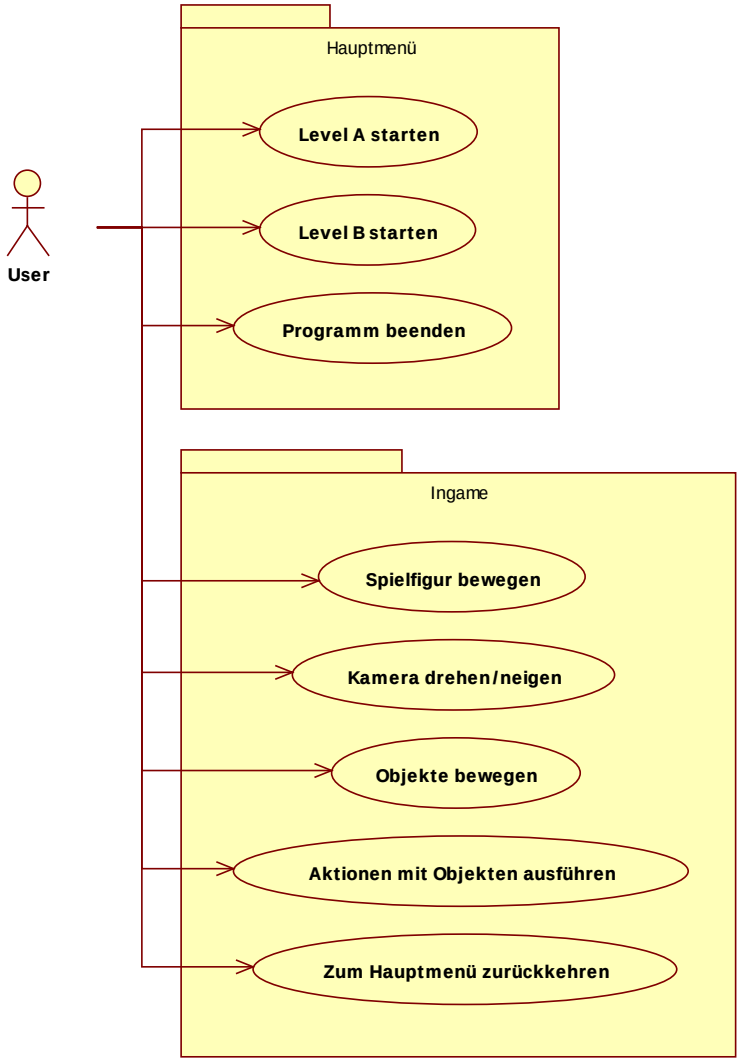

Abbildung 11: Use Case Diagramm

Das Programm ist in zwei Bereiche eingeteilt, das Hauptmenü in dem das Level ausgewählt werden kann. Zudem soll das Spiel im Hauptmenü auch beendet werden können.
Der zweite Bereich ist die Spielansicht, in der der Benutzer die Möglichkeit hat sich zu bewegen und mit Objekten zu interagieren.
In der Spielansicht hat der Benutzer auch die Möglichkeit wieder ins Hauptmenü zurückzukehren.

8.2 Funktionale Anforderungen

8.2.1 Nutzen von funktionalen Anforderungen

"Funktionale Anforderungen beschreiben was das Zielsystem leisten soll. Das Problem von Fließtexten ist, dass Sachverhalte ungenau beschrieben, oder wegen „Offensichtlichkeit“ schlicht weggelassen werden. Um diese Probleme zu vermeiden wird die in **Abbildung 12** sichtbare Schablone nach Rupp benutzt. Die Schablone gewährt eine Möglichkeit Anforderungen klar und eindeutig zu formulieren."**[Jul10]**

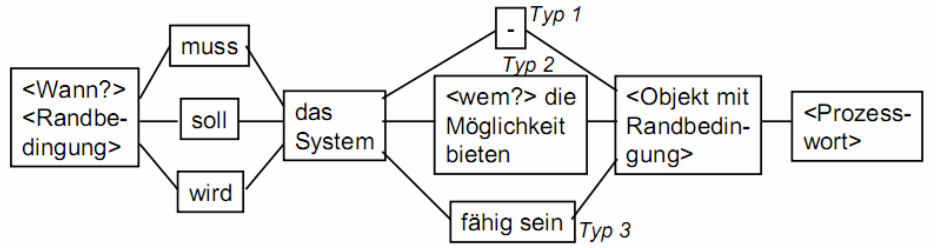

Abbildung 12: Rupp Schablone

"Typ 1: Selbstständige Systemaktivität, d.h. das System führt den Prozess selbständig durch. Ein Beispiel ist die Berechnung des bisherigen Aufwands eines Projekts durch die Abfrage aller Teilprojekte und die Anzeige des Ergebnisses.
Typ 2: Benutzerinteraktion, d.h. das System stellt dem Nutzer die Prozessfunktionalität zur Verfügung. Ein Beispiel ist die Verfügbarkeit eines Eingabefeldes, um Projektdaten einzugeben.
Typ 3: Schnittstellenanforderung, d. h. das System führt einen Prozess in Abhängigkeit von einem Dritten (zum Beispiel einem Fremdsystem) aus, ist an sich passiv und wartet auf ein externes Ereignis. Ein Beispiel ist die Fähigkeit des Systems, die Anfrage einer anderen Bürosoftware nach einer Übersicht über die laufenden Projekte anzunehmen." **[Ste]**

8.2.2 Funktionale Anforderungen Hauptmenü

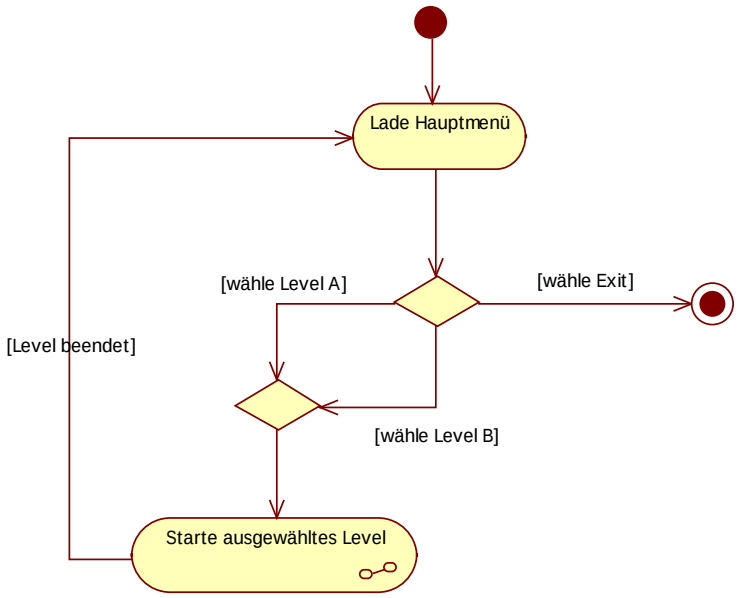

Abbildung 13: Aktivitätsdiagramm - Hauptmenü

HM 1. Nachdem das Programm gestartet wurde, soll es dem Benutzer das Hauptmenü anzeigen.

HM 2. Sobald das Hauptmenü zu sehen ist, soll das System dem Benutzer die Möglichkeit geben Level A zu starten.

HM 3. Sobald das Hauptmenü zu sehen ist, soll das System dem Benutzer die Möglichkeit geben Level B zu starten

HM 4. Sobald das Hauptmenü zu sehen ist, soll das System dem Benutzer die Möglichkeit geben das Programm zu beenden.

HM 5. Sobald der Benutzer ein Level gewählt hat, soll das System mit **IG 1** fortfahren.

8.2.3 Funktionale Anforderungen Ingame

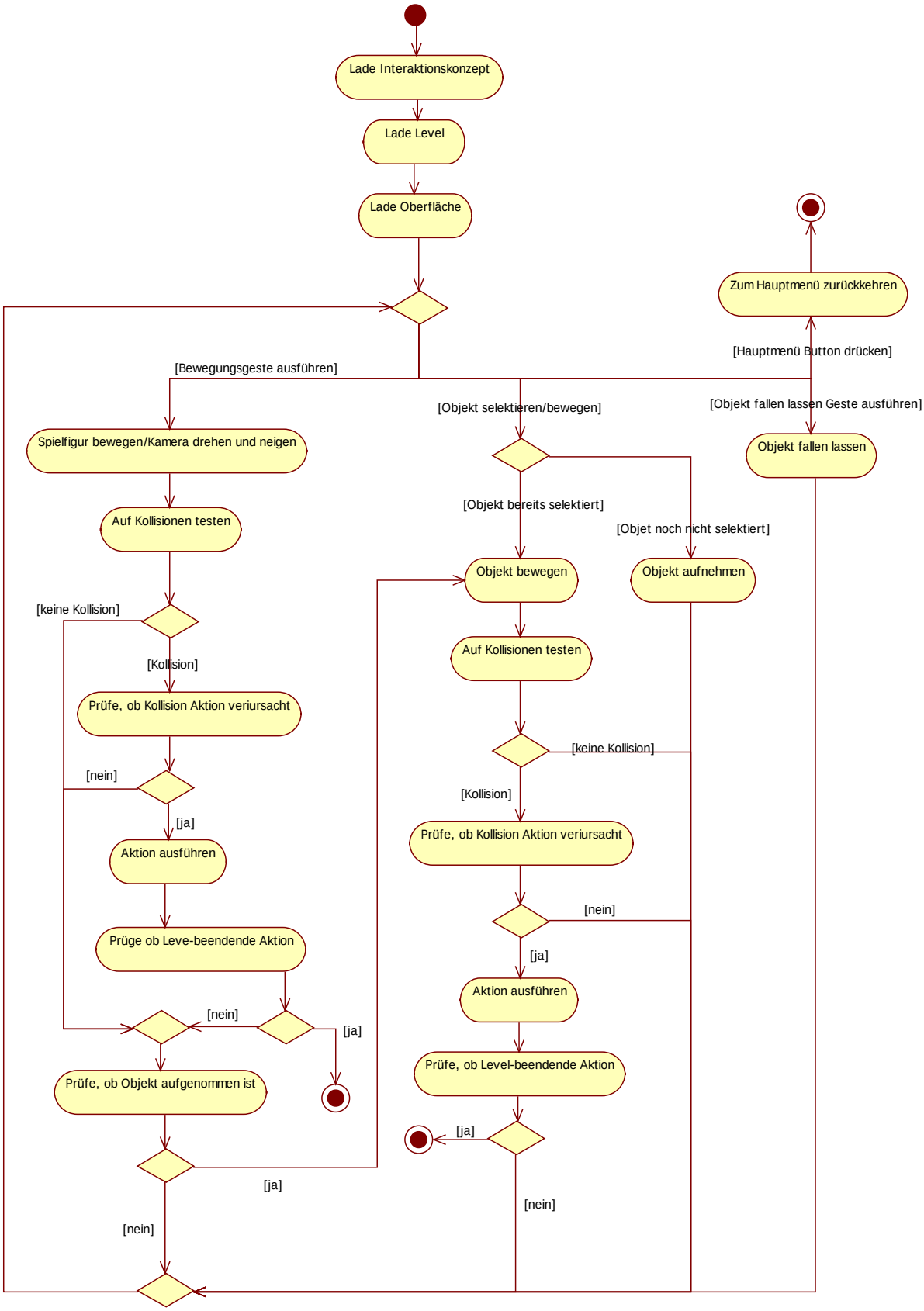

Abbildung 14: Aktivitätsdiagramm - Ingame

IG 1. Sobald ein Level geladen werden soll, soll das System das dazugehörige Interaktionskonzept laden.

IG 2. Nachdem das Interaktionskonzept geladen wurde, soll das System alle Levelinhalte laden.

IG 3. Nachdem alle Levelinhalte geladen wurden, soll das System die Oberfläche anzeigen.

IG 4. Sobald die Oberfläche geladen wurde, soll das System dem Benutzer die Möglichkeit geben Bewegungsgesten auszuführen.

IG 5. Sobald die Oberfläche geladen wurde, soll das System dem Benutzer die Möglichkeit geben Objekte zu selektieren.

IG 6. Sobald die Oberfläche geladen wurde, soll das System dem Benutzer die Möglichkeit geben Objekte fallen zu lassen.

IG 7. Sobald die Oberfläche geladen wurde, soll das System dem Benutzer die Möglichkeit geben zum Hauptmenü zurückzukehren.

IG 8. Sobald der Benutzer eine Bewegungsgeste ausführt, soll das System die Bewegung entsprechend der Geste ausführen.

IG 9. Wurde die Bewegung ausgeführt, soll das System testen, ob die Spielfigur mit einem anderen Objekt kollidiert ist.

IG 10. Hat keine Kollision stattgefunden, soll das System mit Schritt **IG 15** fortfahren.

IG 11. Hat eine Kollision stattgefunden soll das System prüfen, ob die Kollision eine Aktion ausgelöst hat.

IG 12. Wurde durch die Kollision keine Aktion ausgeführt, soll das System mit Schritt **IG 15** fortfahren.

IG 13. Wurde durch die Kollision eine Aktion ausgelöst, soll das System diese Aktion ausführen.

IG 14. War die Aktion eine Level beendende Aktion, soll zum Hauptmenü zurückgekehrt werden (Einstieg bei Schritt **HM 1**).

IG 15. Wurde keine Kollision festgestellt, oder falls die Kollision keine Aktion verursacht hat, oder wenn die Aktion keine Level beendende Aktion war, soll das System prüfen, ob der Spieler ein Objekt bei sich trägt.

IG 16. Trägt der Spieler kein Objekt bei sich, soll nach Schritt **IG 3** fortgefahren werden.

IG 17. Trägt der Spieler ein Objekt bei sich, soll bei Schritt **IG 21** fortgefahren werden.

IG 18. Sobald der Benutzer ein Objekt selektieren/bewegen möchte, soll das System prüfen, ob das Objekt bereits selektiert war.

IG 19. War das Objekt noch nicht selektiert, soll das System das Objekt als selektiert kennzeichnen.

IG 20. Wurde das Objekt als selektiert gekennzeichnet, soll nach Schritt **IG 3** fortgefahren werden.

IG 21. War das Objekt bereits selektiert, soll das System das Objekt bewegen.

IG 22. Wurde das Objekt bewegt, soll das System auf Kollisionen mit anderen Objekten testen.

IG 23. Hat keine Kollision stattgefunden, soll das System mit Schritt **IG 3** fortfahren.

IG 24. Hat eine Kollision stattgefunden soll das System prüfen, ob die Kollision eine Aktion ausgelöst hat.

IG 25. Wurde durch die Kollision keine Aktion ausgeführt, soll das System mit Schritt **IG 3** fortfahren.

IG 26. Wurde durch die Kollision eine Aktion ausgelöst, soll das System diese Aktion ausführen.

IG 27. War die Aktion eine Level beendende Aktion, soll zum Hauptmenü zurückgekehrt werden (Einstieg bei Schritt **HM 1**).

IG 28. War die Aktion keine Level beendende Aktion soll mit Schritt **IG 3** fortgefahren werden.

IG 29. Hat der Benutzer die Geste zum fallen lassen des Objekts ausgeführt, soll das System das Objekt fallen lassen.

IG 30. Wurde das Objekt fallen gelassen, soll das System nach Schritt **IG 3** fortgefahren.

IG 31. Hat der Benutzer den Button zum Verlassen des Spiels gedrückt, soll das System bei Schritt **HM 1** fortfahren.

9 Feinkonzept

Im Feinkonzept wird die Modellierung der Software genauer erläutert. Es werden die Zusammenhänger der Softwarepakete erklärt, sowie die der Klassen in den Paketen.

9.1 Paketdiagramm

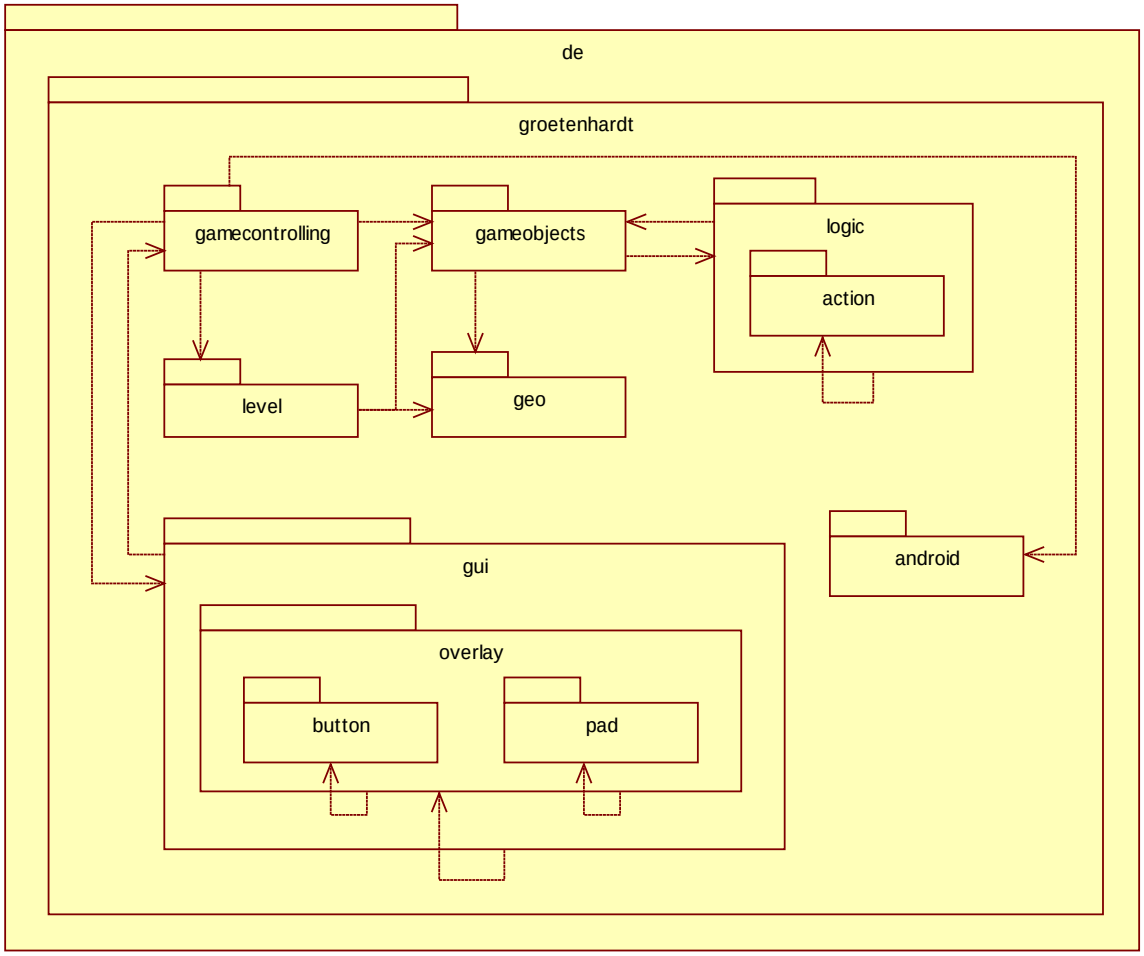

Abbildung 15: Paketdiagramm

Dieses *Paketdiagramm* stellt die Abhängigkeiten der erstellten Pakete dar. Vorerst wird erläutert welche Funktionen die einzelnen Pakete haben, anschließend werden die Abhängigkeiten erklärt.

de.groetenhardt.gui

Beinhaltet alle Klassen, die zur Darstellung des Programms dienen. Dazu gehören die Activity zur Anzeige der 3D-Umgebung, sowie verschiedene Interfaces, die benötigt werden, um auf die Benutzeroberfläche Einfluss zu nehmen.

Dieses Paket benutzt das Unterpaket "overlay", um das 2D Overlay erzeugen und anzeigen zu können.
Des Weiteren werden hier die Benutzereingaben angenommen und an das Gamecontrolling weitergeleitet.

de.groetenhardt.gui.overlay
Dieses Paket beinhaltet alle Klassen, die zur Darstellung des 2D Overlays des Spiels benötigt werden, darunter ebenfalls verschiedene Interfaces, die benötigt werden, um Einfluss auf das Overlay zu nehmen.
Das Paket hat Zugriff auf die Unterpakete "button" und "pad", um diese Steuerelemente darstellen zu können.

de.groetenhardt.gui.overlay.button
Beinhaltet die Logik der Buttons, die auf dem Overlay dargestellt werden können.

de.groetenhardt.gui.overlay.pad
Stellt die Logik des virtuellen Pads dar, das für das Konzept "vPad" benötigt wird.

de.groetenhardt.gamecontrolling
In diesem Paket befinden sich einige Klassen, die für die Einhaltung der Spielregel zuständig sind. Hier wird beispielsweise die Steuerung durch den Benutzer umgesetzt, sowie die Kollisionserkennung angestoßen.
Auch die beiden Interaktionskonzepte werden als Klassen in diesem Paket umgesetzt.
Das Gamecontrolling hat Zugriff auf die Benutzeroberfläche, um die Eingaben des Benutzers visuell übertragen zu können.
Auch der Zugriff auf die Spielelemente aus dem Paket "gameobjects", sowie auf das Paket "level" wird dem Gamecontrolling gestattet. Im Paket "gameobjects" befindet sich beispielsweise die Klasse, die den Spieler symbolisiert, welcher durch das Gamecontrolling gesteuert wird.
Aus dem Paket "level" werden alle Levelinhalte (wie Gegenstände, Wände und Böden) bezogen, um sie in den Spielregeln beachten zu können.

de.groetenhardt.level
Hier können Level von einer Abstrakten Klasse abgeleitet werden. Die Level sind lediglich eine Ansammlung aller Objekte, die in der Spielwelt zum Einsatz kommen. Spielerische Logik wird hier nicht realisiert.

de.groetenhardt.gameobjects

Dieses Paket beinhaltet, alle Objekte, die zum Leveldesign zur Verfügung stehen. Die Spielobjekte beinhalten unter anderem die Logik, die nötig ist, um bewegt und selektiert zu werden.

Die Spielobjekte erhalten Zugriff auf das Paket "geo", welches Elemente enthält, aus denen die Objekte zusammengesetzt sind.

de.groetenhardt.logic

In diesem Paket sind die Klassen zur Erstellung des Spielablaufs zu finden. Beispielsweise gibt es hier die Klasse "Interaction" mit der es möglich ist Aktionen auf bestimmte Ereignisse zu legen. Aktionen können aus dem Unterpaket "action" bezogen werden, die es beispielsweise erlauben Texte anzuzeigen, Objekte zu transformieren, oder das Spiel zu beenden.

de.groetenhardt.geo

Hier befinden sich Klassen, um die Geometrie realisieren zu können. Beispielsweise ist hier die Klasse zu finden, die es ermöglicht Figuren in OpenGL ES zu erstellen.

de.groetenhardt.android

Enthält eine Klasse, die Zugang zur Hardware des Smartphones verschafft. Beispielsweis kann hier eine Vibration ausgelöst werden.

9.2 Klassendiagramm

Das *Klassendiagramm* kann aufgrund seiner Größe leider nicht als Ganzes dargestellt werden, daher werden nur erwähnenswerte Ausschnitte gezeigt und beschreiben.

Das komplette Klassendiagramm kann als PDF auf der Thesis CD betrachtet werden.

9.2.1 Modellierung der Interaktionskonzepte

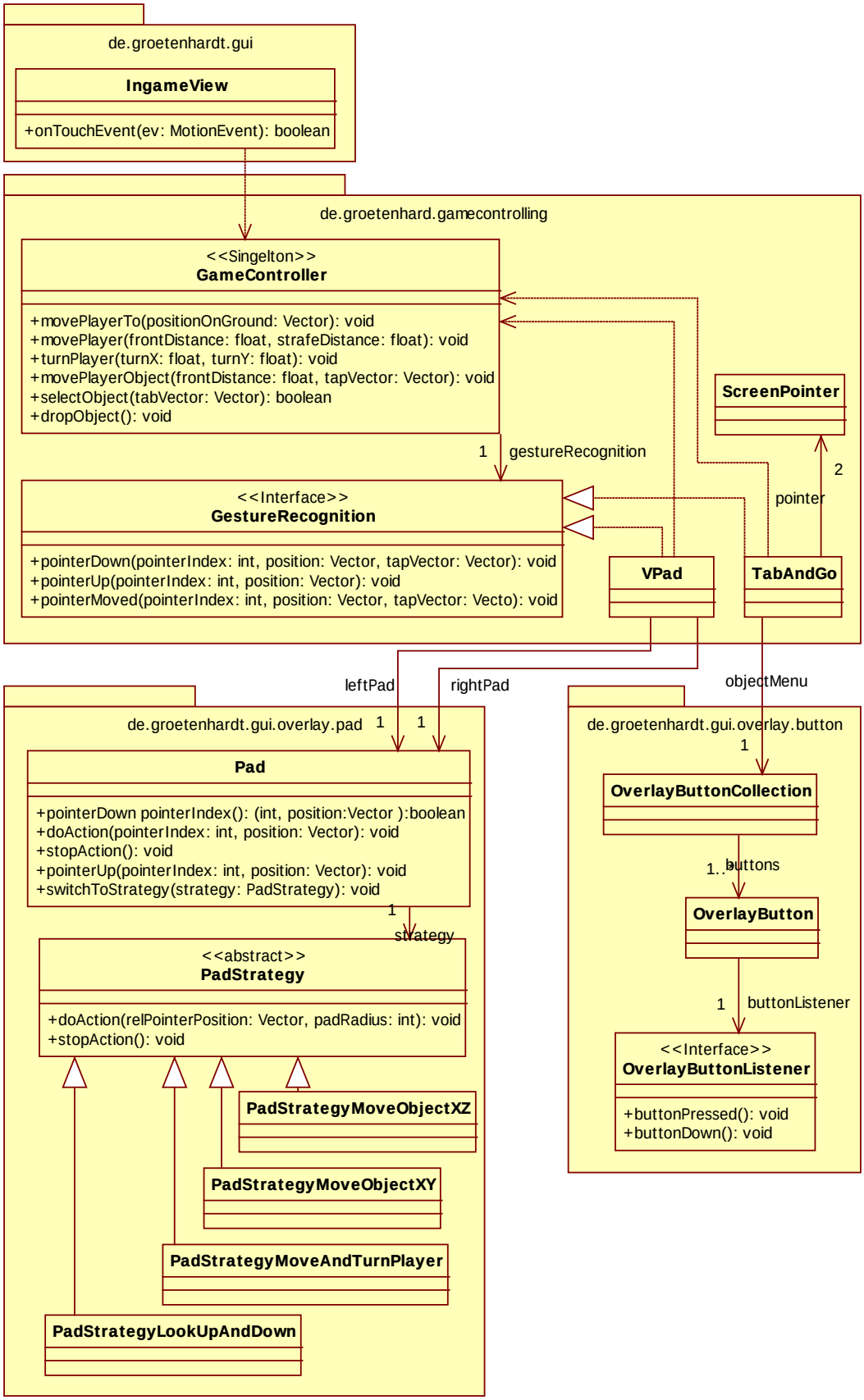

Abbildung 16: Ausschnitt Klassendiagramm - Modellierung der Interaktionskonzepte

In **Abbildung 16** ist der Ausschnitt des Klassendiagramms zu sehen, welcher zeigt wie die Interaktionskonzepte in der Software eingebettet sind. Alle andern Klassen und Verbindungen zu ihnen sind ausgeblendet.
Der "GameController" besitzt ein Interface "GestureRecognition", welchem mitgeteilt wird welche Aktionen der Benutzer gerade auf dem Display ausübt, welcher Finger sich auf dem Screen befindet und welcher Finger sich bewegt. Die Benutzeraktionen werden vom "IngameView" übermittelt.
Dieses Interface wird von den Klassen "VPad" und "TabAndGo" implementiert.

"VPad" stellt, wie der Name schon sagt, das Interaktionskonzept vPad dar, welches die Benutzereingabe insofern verarbeitet, dass festgestellt wird welches der beiden "Pad"s in die Aktion involviert ist und ob durch die Aktion der Bewegungsmodus umgeschaltet werden muss (Bewegungs- und Objektbewegungsmodus vergleich **5.3 vPad**). Ist eines der Pads mit der Aktion des Benutzers angesprochen worden, wird diese je nach Strategie des Pads ausgewertet und an den "GameController" zurückgegeben.
Die Strategien werden verwendet, um die Pads so modular wie möglich zu gestalten, das heißt, dass die Pads ohne Probleme eine andere Strategie im laufenden Betrieb annehmen können.
Dies ermöglicht es beispielsweise zuerst mit einem Pad den Spieler zu steuern und später, wenn ein Objekt bewegt werden soll, kann diese Bewegung mit demselben Pad durchgeführt werden, indem die Strategie gewechselt wird.

Die Klasse "TabAndGo" stellt das Interaktionskonzept TabAndGo dar, welches auf Eingabehilfen wie das "Pad" verzichtet. Stattdessen wertet es alle Bewegungen direkt aus und gibt sie an den "GameController" weiter. Nur die "ScreenPointer" helfen "TabAndGo", indem sie die letzte Position der Finger speichern und dadurch eine Geschwindigkeit ableiten können. Zudem hat es Zugriff auf eine "ButtonCollection", welche ein Menü darstellt, das erscheint, wenn ein Objekt lange mit einem unbewegten Finger berührt wird.

9.2.2 Modellierung der Spielaktionen

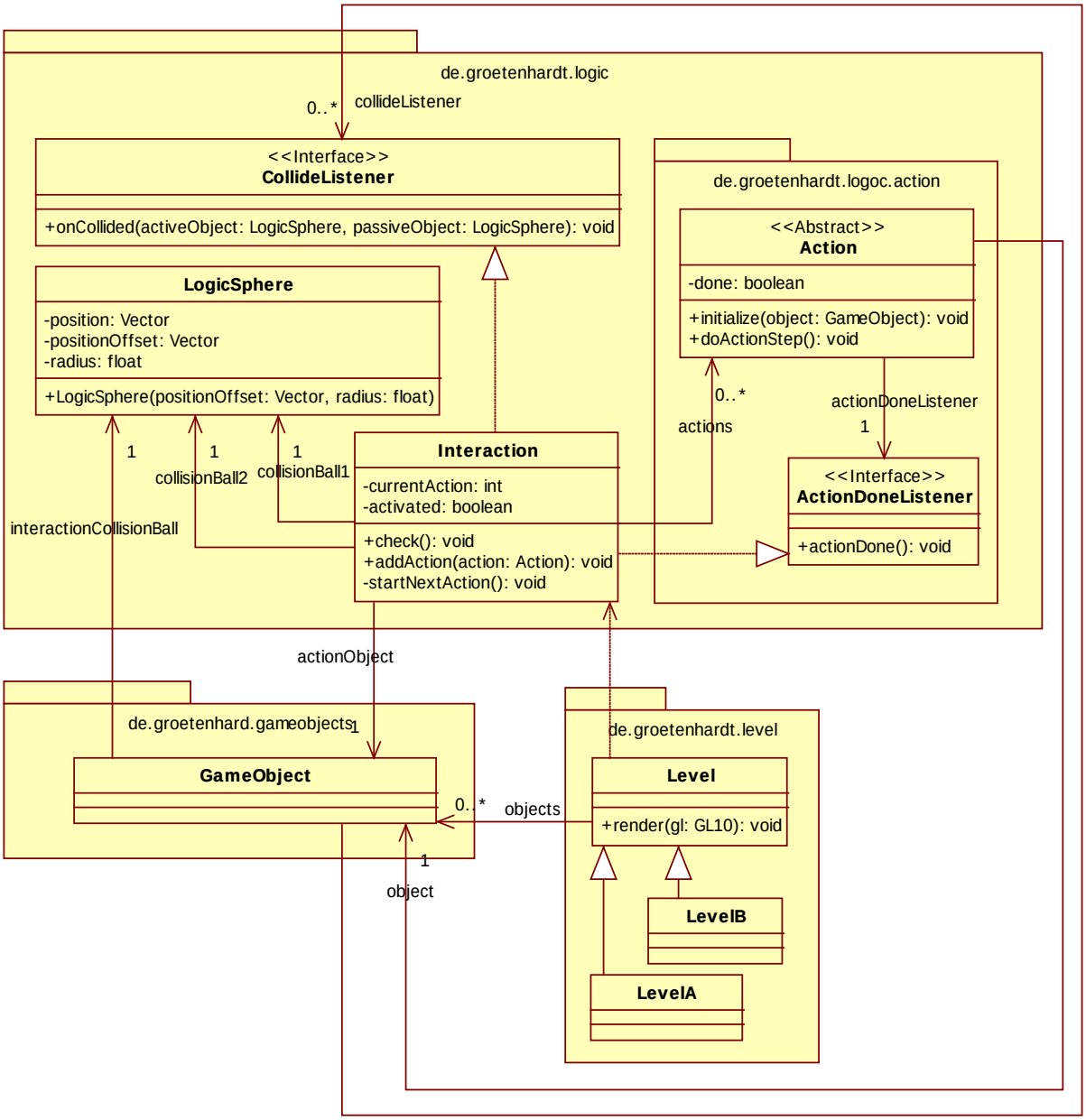

Abbildung 17: Ausschnitt Klassendiagramm - Modellierung der Spielaktionen

Zu sehen ist in **Abbildung 17** der Ausschnitt des Klassendiagramms, welcher zeigt wie die Spielaktionen modelliert wurden. Unter Spielaktionen wird in diesem Kontext verstanden, dass durch eine Kollision zweier in der Spiellogik verbundener Objekte eine Aktion ausgelöst wird, beispielsweise soll bei der Kollision eines Schlüssels mit einer Tür diese durch eine Rotation geöffnet werden. Die Kollision selber wird in diesem Kontext als Interaktion und das Rotieren als Aktion bezeichnet.

Die Klasse "Level" beinhaltet alle Spielelemente wie Wände, Böden und sonstige Objekte, wie Türen oder Schlüssel. Die sonstigen Objekte sind im Diagramm als "GameObject"-Klasse dargestellt.
"GameObject"s sind Objekte, mit denen der Spieler im Spiel interagieren kann, das heißt er kann sie bewegen und Aktionen mit ihnen auslösen. Wie das genau geschieht, wird im Folgenden erläutert.

Die Klasse "Interaction" wird von "Level" verwendet, um zwei Spielobjekte logisch miteinander zu verknüpfen. "Interaction" benötigt dazu von beiden Spielobjekten jeweils eine Kollisionskugel, die Member Variable "interactionCollisionBall" des "GameObject"s (vergleich **10.2.1 Objektaufbau im Projekt**). Zudem bekommt das Spielobjekt einen "CollideListener", mit dem die "Interaction" über Kollisionen der beiden Objekte informiert wird.
Kollidieren zwei durch eine "Interaction" verbundene Objekte mit ihren "interactionCollisionBall"s, wird in "Interaction" die Methode "startNextAction()" ausgelöst, angestoßen durch den "CollideListender" des betroffenen "GameObject"s.
"startNextAction()" startet die nächste anstehende "Action". Die "Action" realisiert die Aktion, die mit dem angegebenen Spielobjekt geschehen soll. Beispielsweise kann eine "Action" ein Objekt rotieren oder verschieben, sogar das Gewinnen eines Levels ist durch eine solche "Action" realisierbar. Ist eine Aktion vollendet, wird dies der "Interaction"-Instanz anhand des "ActionDoneListeners" mitgeteilt. Diese wiederum startet die nächste "Action", falls vorhanden.

9.2.3 Modellierung der Benutzeroberfläche

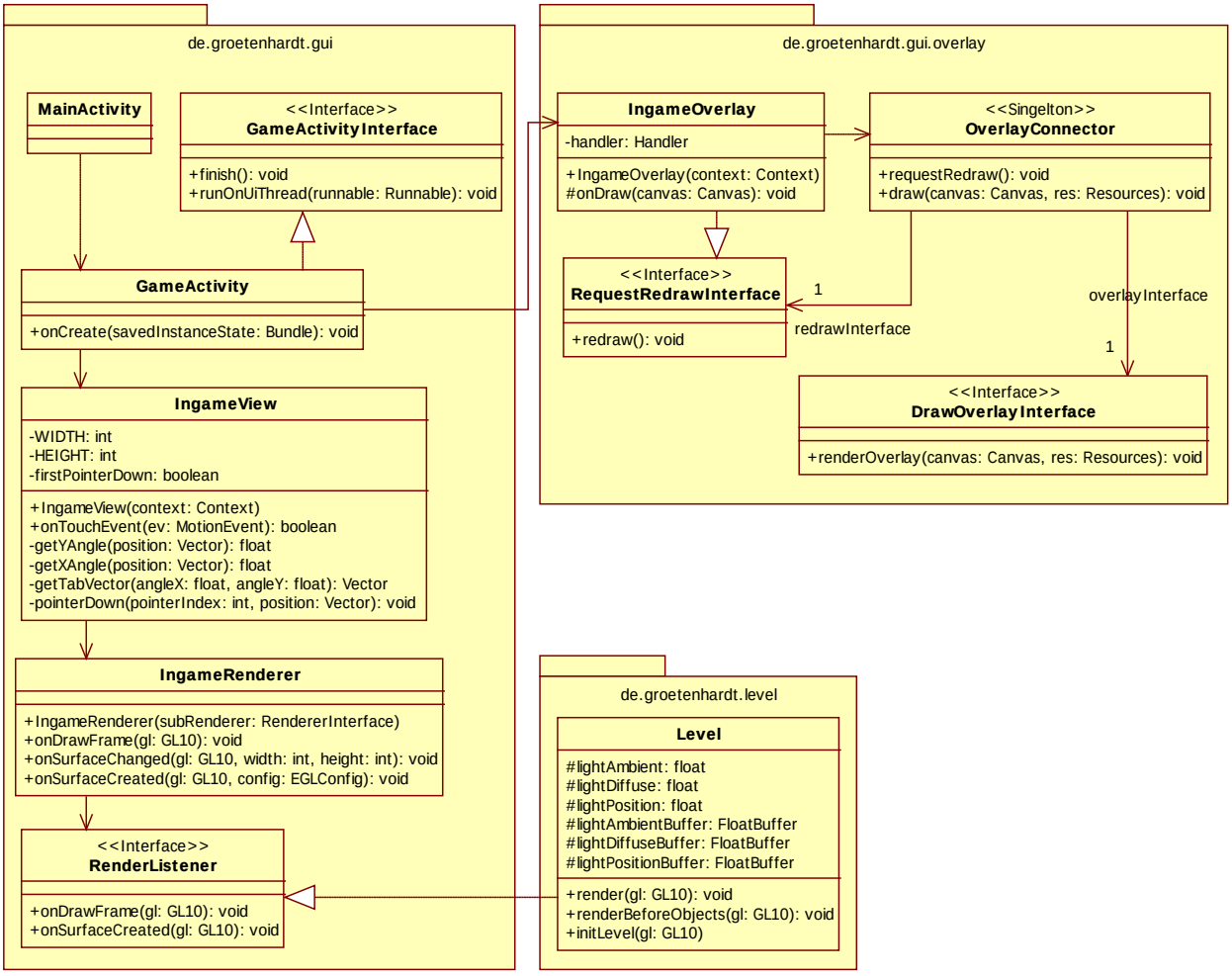

Abbildung 18: Ausschnitt Klassendiagramm - Modellierung der Benutzeroberfläche

Der in **Abbildung 18** zu sehende Ausschnitt des Klassendiagramms zeigt den Aufbau der Benutzeroberfläche.
Das Spiel startet mit der "MainActivity", welche das Hauptmenü anzeigt. Wird ein Level gewählt, wird die "GameActivity" gestartet. Die "GameActivity" besitzt zwei Views, den "IngameView" und den "IngameOverlay"-View.

"IngameView" erbt von GLSurfaceView, welcher es ermöglicht OpenGL Grafiken anzuzeigen. "IngameRenderer" ist dafür zuständig die OpenGL Befehle auszuführen.
Dazu besitzt er einen Hilfsrenderer, das "RenderListener"-Interface, welches vom "Level" implementiert wird. Das "Level" enthält alle Elemente, die gezeichnet werden können, stößt also die meisten OpenGL-Befehle an.
Im "IngameRenderer" werden nur die OpenGL Grundeinstellungen vorgenommen, wie beispielsweise das setzen von glViewport und gluPerspective.

Das "IngameOverlay" erbt von View, hat also keine besonderen Eigenschaften. Es wird lediglich benötigt ein 2D Benutzerinterface darzustellen.
Es Implementiert das "RequestRedrawInterface", welches vom "OverlayConnector" genutzt wird, um einen Redraw (das neue Zeichnen der betroffenen Elemente) anzufordern. Ein Redraw wird immer dann angestoßen, wenn sich die 2D Oberfläche verändert hat. Eine Veränderung wäre beispielsweise die Bewegung eines Steuerknüppels des vPads. Daher ist es den Interaktionskonzepten möglich über den "OverlayConnector" ein Redraw anzufordern. Zudem sind die Interaktionskonzepte beim "OverlayConnector" als "DrawOverlayInterface" angemeldet, um die Recourcen zum Zeichnen ihrer 2D Elemente zu bekommen.

10 Theoretische Problemlösungen

In diesem Abschnitt werden interessante Probleme vorgestellt und beschrieben, wie diese gelöst wurden.

10.1 Objektselektion

Da in dem Spiel die Möglichkeit gegeben sein soll mit Objekten zu interagieren, muss dem Spieler die Möglichkeit gegeben werden ein Objekt zu selektieren, mit dem er interagieren möchte. Um ein Objekt zu selektieren, soll es in der fertigen Software lediglich mit dem Finger berührt werden.

Das Problem ist, dass nur die 2D Koordinate der Berührung auf dem Display bekannt ist. Von dieser Koordinate muss auf ein Objekt in einer 3D Umgebung geschlossen werden. Eine 2D Koordinate kann auf eine 3D Welt als Strahl übertragen werden. Das heißt: Drückt der Spieler auf das Display, geht ein Strahl von der Kameraposition durch den Punkt der Projektionsebene, der von dem Spieler berührt wurde, wie in **Abbildung 19** abstrakt dargestellt.

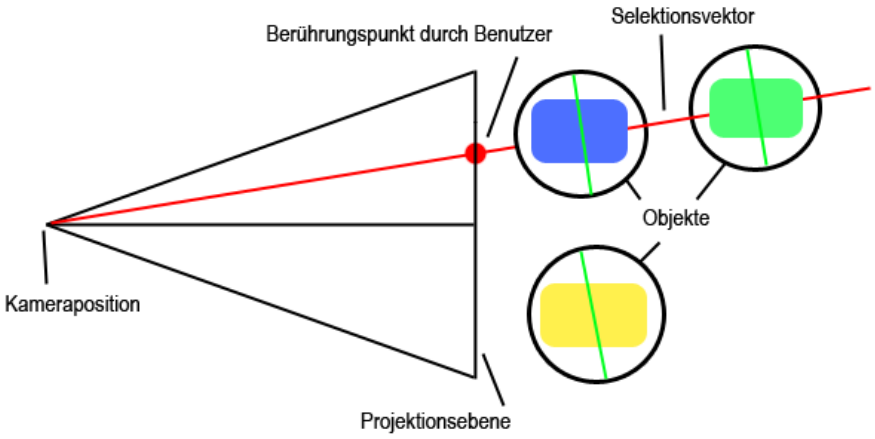

Abbildung 19: Skizze zur Objektselektion

Der Strahl wird berechnet, indem die Position auf dem Display als Winkel relativ zum Mittelpunkt des Displays berechnet wird. Im Programm ist festgelegt, dass das vertikale Sichtfeld 45° beträgt. Das horizontale Sichtfeld wird relativ zur Displaybreite bestimmt, im Fall des Samsung Galaxy S I9000 sind das 75°. Die Displaybreite umfasst also 75° und die Displayhöhe 45°. Das heißt es kann mit einem simplen Dreisatz bestimmt werden welchen Winkel zur x- bzw. y-Achse der Berührungspunkt zum Displaymittelpunkt hat.
Diese Winkel werden auf den Blickvektor in der Welt übertragen, dadurch wird der tatsächlich Strahl berechnet.

Zudem ist in **Abbildung 19** ein weiteres Problem dargestellt, denn der Strahl kann auch durch mehrere Objekte verlaufen (in diesem Fall durch das blaue und das grüne Objekt). Das bedeutet, dass entschieden werden muss welches der beiden Objekte gemeint ist. Im Normalfall soll das Objekt selektiert werden, das näher am Spieler ist.

Ein weiteres Problem ist: Wie bekommt man heraus, ob das Objekt überhaupt durch den Strahl geschnitten wurde? Dazu erhält das Objekt (wie in der Abbildung angedeutet) eine Kugel als Mantel. Dies wird praktiziert, da die Feststellung eines Treffers wesentlich einfacher ist, als würde jede Fläche des Objekts auf einen Schnitt überprüft werden.
Es wird eine virtuelle Fläche auf die Position des Objektmittelpunktes gelegt, welche eine Normale hat, die dem Vektor des Strahls entspricht (in der Abbildung durch die grünen Striche, die die Objekte schneiden angedeutet). Als nächstes wird der Schnittpunkt des Strahls mit der virtuellen Fläche berechnet, ist die Entfernung des Schnittpunkts zur Objektmitte kleiner als der Radius des Mantels, kommt das Objekt für eine Selektion in Frage.

Im Laufe der Implementierung wurde die Selektion etwas verfeinert, da nun mit den Kollisionskugeln und -flächen (wird in **10.2.1 Objektaufbau im Projekt** erläutert) gearbeitet wird, das Prinzip bleibt allerdings dasselbe.

10.2 Kollisionserkennung

Einige Informationen zur Kollisionserkennung aus den Videos von **[Bil11]** .

Ein weiteres Problem, das im Spiel gelöst werden muss, ist die Kollisionserkennung. Diese wird zum einen benötigt, um Aktionen hervorzurufen, wenn zwei Objekte miteinander kollidieren (z.B. Schlüssel trifft Schlüsselloch). Zum anderen darf der Spieler seine vorgegebene Welt nicht verlassen, muss also gegen Wände stoßen können.

Die Kollisionserkennung spielt in Spielen, in denen sie benötigt wird, eine große Rolle. Zum einen darf sich nicht zu rechenaufwändig sein, zum anderen muss sie genau genug sein, um das Spiel logisch erscheinen zu lassen.
Es gibt einige Tricks den Rechenaufwand der Kollisionserkennung gering zu halten. Beispielsweise können um Objekte Kugeln gelegt werden, welch auf Kollisionen geprüft werden, Kollisionen von Kugeln sind sehr leicht zu berechnen und gehen daher sehr schnell, dadurch kann eine Vorauswahl getroffen werden, ob das Objekt für eine genauere Kollisionserkennung in Frage kommt. Je nach Spiel reicht sogar eine Kollisionsüberprüfung des virtuellen Mantels schon aus.

Aber auch schon vorher kann eine Auswahl getroffen werden, welche Objekte auf Kollisionen getestet werden. Eine Möglichkeit ist hierbei das Level in Sektoren zu unterteilen. Das erleichtert die Kollisionserkennung insofern, dass nur Objekte innerhalb der Sektoren auf Kollisionen getestet werden müssen. Beispielsweise müsste dann Objekt 1 welches sich in Sektor A befindet nicht auf Kollisionen mit Objekt 2 aus Sektor B getestet werden.
Dies ist aber nur ein Beispiel und wird aufgrund der geringen Komplexität des Spiels in diesem Projekt nicht praktiziert.

10.2.1 Objektaufbau im Projekt

Was nun folgt ist eine genauere Beschreibung der Kollisionserkennung, wie sie in diesem Projekt angewendet wird.

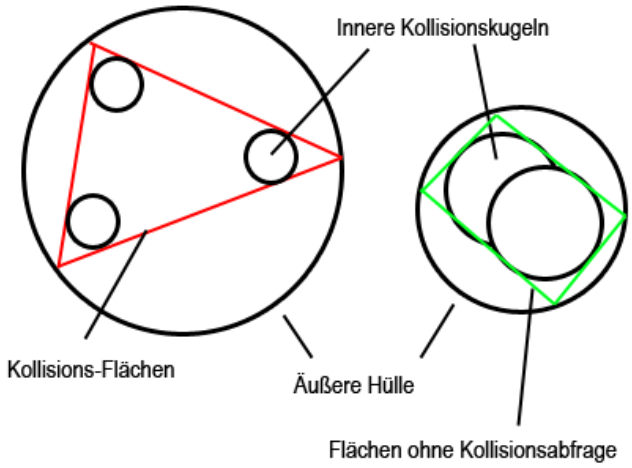

Abbildung 20: Skizze zu den logischen Kollisionseinheiten

Abbildung 20 zeigt wie die Objekte in diesem Spiel aufgebaut sind. Alle Objekte haben eine äußere Hülle, die zur Vorauswahl der zu prüfenden Objekte dient. Kollidieren die äußeren Hüllen der Objekte, wird eine feinere Kollisionserkennung durchgeführt.
Des Weiteren haben alle Objekte innere Kollisionskugeln und Flächen, mit denen eine Kollision möglich ist, sowie Flächen mit denen keine Kollision möglich ist (Flächen, die nur angezeigt werden).
Die beiden unterschiedlichen Flächentypen werden eingesetzt, da bei großen Flächen (z.B. bei Türen) eine Kollisionsabfrage sinnvoll ist. Bei kleineren Flächen, die sehr nahe bei einer der inneren Kollisionskugeln liegen, ist eine Abfrage nicht nötig, da die Art von Ungenauigkeit akzeptiert werden kann.
Auf Kollisionen kann in diesem Projekt zwischen Kugeln und Kugeln, sowie zwischen Kugeln und Flächen getestet werden, nicht aber zwischen Flächen und Flächen. Da auf Kollisionen zwischen Flächen nicht getestet werden kann, benötigt ein Objekt, das nur aus Kollisionsflächen bestehen könnte dennoch innere Kollisionskugeln. In diesem Fall werden

Kollisionskugeln an allen Ecken benötigt, um nicht durch andere Fläche hindurch bewegt werden zu können.

Dieser Sachverhalt kann an den in **Abbildung 20** zu sehenden Objekten gezeigt werden. Das Rechteck kann auf kollidierbare Flächen verzichten, da es fast komplett von den inneren Kollisionskugeln ausgefüllt wird. Die Ecken, die möglicherweise in einem anderen Objekt verschwinden könnten sind vernachlässigbar.
Das Dreieck hingegen darf nicht auf kollidierbare Flächen verzichten, da die inneren Kollisionskugeln das Objekt nicht genug ausfüllen. Das Rechteck könnte zum Beispiel ohne die Kollisionsflächen problemlos in das Dreieck eindringen.
Wären nur diese beiden Objekte existent, könnte sogar auf die Kollisionskugeln des Dreiecks verzichtet werden, da die Kollisionsflächen das Eindringen des Rechtecks verhindern würden.
Gäbe es allerdings zwei dieser Dreiecke ohne Kollisionskugeln, könnten diese ungehindert ineinander eindringen, da Flächen nicht auf Kollisionen mit Flächen getestet werden. Deshalb sind die Kollisionskugeln unabdingbar.

10.2.2 Die Kollisionserkennung

Nun zur Methode wie eine Kollision festgestellt wird. Soll eine Bewegung ausgeführt werden, ist die alte und neu Position bekannt. Das heißt es muss überprüft werden, ob durch die Bewegung zur neuen Position eine Berührung mit einem anderen Objekt entsteht, was durch die Entfernung der Kollisionskugel und einer anderen Kollisionskugel oder Fläche bestimmt wird.
Würden sich die Objekte nach der Bewegung berühren, soll die Bewegung verhindert werden.

Wird allerdings nur mit der neuen Position auf eine Kollision getestet, entsteht das in **Abbildung 21** skizzierte Problem. Es kann sein, dass die Bewegung so extrem ist, dass zwischen alter und neuer Position ein Freiraum entsteht, in dem sich beispielsweise eine Fläche befindet, die eigentlich nicht durchdrungen werden darf.

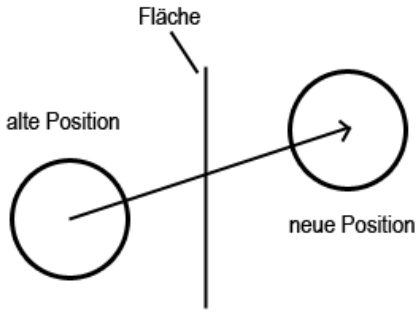

Abbildung 21: Skizze zur Kollisionserkennung

Um dieses Problem zu lösen, wird die Kollisionserkennung iterativ durchgeführt. Das bedeutet, dass die aktuelle Kollisionskugel stückweise in Bewegungsrichtung versetzt wird und auf Kollisionen getestet wird, wie in **Abbildung 22** dargestellt. Sobald in einem der Schritte eine Kollision festgestellt wird, kann die Überprüfung abgebrochen werden. Die Schrittweite ist hier durch den Radius der Kugel definiert. Als Schrittweite wurde der Radius gewählt, da dadurch ausreichen viel abgetastet wird. Würde der Durchmesser als Schrittweite verwendet werden, könnte es sein, dass die Kugel durch ein winziges Loch in einer Wand passt, wenn diese genau zwischen den Kollisionskugeln zweier Iterationen liegt.

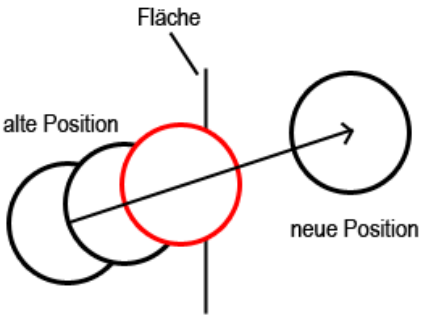

Abbildung 22: Skizze zur iterativen Kollisionserkennung

Die gleiche Methode wird nicht nur mit Flächen sondern auch mit Kollisionskugeln durchgeführt.

11 Implementierung

In diesem Abschnitt werden nennenswerte Implementierungen vorgestellt. Uninteressante oder unübersichtliche Codeabschnitte werden mit `[...]` ersetzt, um Platzt zu sparen, oder die Übersichtlichkeit zu erhalten. Zudem werden die Codekommentierungen ebenfalls entfernt, da der Code hier separat erläutert wird.
Bestimmte Codeabschnitte werden mit Nummern markiert, auf die im darauf folgenden Text Bezug genommen wird.

11.1 Umsetzung der Gesten am Beispiel von TabAndGo

Das Interaktionskonzept TabAndGo erhält die meisten Benutzereingaben durch das Multitouch-Display. Diese Benutzereingaben werden mit Hilfe der Klasse `TabAndGo` ausgewertet. Zur Auswertung stehen drei Methoden zur Verfügung `pointerDown`, `pointerUp` und `pointerMoved`.
Der `IngameView` bekommt durch das Android Betriebssystem mitgeteilt, welche Aktionen auf dem Display getätigt werden, welcher sie an `TabAndGo` mit Hilfe der drei Methoden weiterleitet.

11.1.1 pointerDown Methode

```
public void pointerDown(int pointerIndex, Vector position, Vector
            tabVector) {

      [...]

/*1*/ if (pointerIndex == 0) {

            lastLongpressPosition = position;

/*2*/       if (!playerObjectMovement) {
                  playerObjectMovement = GameController.getInstance()
                              .selectObject(tabVector);
            }

/*3*/       GameController.getInstance().setObjectFingerOffset(
                        tabVector);

/*4*/       if ([...]) {

                  Vector tempTab =
                              Math3D.getPointOnArea(Player.getInstance()
                              .getPosition(), tabVector, new Vector(0,0,0)
                              , new Vector(0,1,0));

                  GameController.getInstance().movePlayerTo(tempTab);
            }

/*5*/       if (!pointer.get(1).isOnScreen()) {
```

```
                    lastLongpressPosition = position;
                    distanceSinceLongpressStart = 0;
                    this.startTimer();

                    [...]
                }

                [...]
        }

        pointer.get(pointerIndex).update(true, position);
}
```

Code 3: Ausschnitt der Methode pointerDown aus der Klasse TabAndGo

Die erste Methode ist `pointerDown`, welche den `pointerIndex` (0 = erster Finger, 1= zweiter Finger), die Position des Fingers auf dem Display (`position`), sowie den `tabVector` vom `IngameView` übergeben bekommt. Der `tabVector` ist der Strahl, der in der 3D-Welt erzeugt wird, wenn der Benutzer den Bildschirm berührt (vergleich **10.1 Objektselektion**).
Die Methode bestimmt welche Geste begonnen werden könnte.

1: Untersucht wird in dieser Methode nur die Aktionen des ersten Fingers, da die des zweiten Fingers nur von Bedeutung sind, wenn dieser bewegt wird.

2: Ist noch kein Objekt aktiv, wird überprüft, ob durch diese Berührung eine Objekt selektiert wurde. Die Selektion führt der `GameController`, angestoßen durch die Methode `selectObject`, durch.

3: Da es möglich ist das Spielobjekt zu verschieben, ohne dass sich der erste Finger direkt darauf befindet, muss dem `GameController` mitgeteilt werden wie weit das Spielobjekt und der `tabVector` auseinander liegen. Dies wird in der Methode `setObjectFingerOffset` bestimmt, indem die Winkel (um x- und y-Achse) zwischen `tabVector` und relativer Position des Spielobjekts zum Spieler berechnet werden.

4: Die Bedingung wurde ausgeblendet, da sie etwas unübersichtlich ist. Sie testet, ob ein *Doubletap* stattgefunden hat. Dazu muss ein Finger zwei Mal kurz hintereinander das Display an ungefähr der geleichen Stelle berührt haben.
Hat ein Doubletap stattgefunden, wird berechnet an welcher Stelle der `tabVector` den Boden der 3D-Welt geschnitten hat. an diese Stelle soll der Spieler bewegt werden.
Die Bewegung wird über den Aufruf `movePlayerTo` am `GameController` gestartet.

5: Ist der erste Finger der einzige auf dem Display, ist ein *Longpress* möglich. Dieser wird an dieser Stelle vorbereitet, indem ein Timer gestartet wird, welcher nach kurzer Zeit die Logik des Longpress ausführt. Der Timer wird allerdings wieder beendet, wenn der Finger sich vom Display entfernt, oder auf dem Display eine zu lange Strecke zurücklegt. Zur Berechnung der zurückgelegten Fläche dient die Member Variable `distanceSinceLongpressStart.`

11.1.2 pointerUp Methode

```
public void pointerUp(int pointerIndex, Vector position) {

/*1*/ longPress.cancel();

/*2*/ if (pointerIndex == 0) {

            [...]

            if (!pointer.get(1).isOnScreen()) {

                  playerObjectMovement = false;
            }

      } else {

            if (!pointer.get(0).isOnScreen()) {
                  playerObjectMovement = false;
            }
      }

/*3*/ pointer.get(pointerIndex).update(false, position);
}
```

Code 4: Ausschnitt der Methode pointerUp aus der Klasse TabAndGo

Die `pointerUp` Methode bekommt den `pointerIndex`, sowie die `position` auf dem Display übergeben. Sie wird hauptsächlich benötigt, um Gesten zu unterbrechen.

1: Wird ein Finger vom Display entfernt, wird der Longpress unterbrochen.

2: Dieser if-Block entscheidet, ob der Objektbewegungsmodus beendet wird. Er wird beendet, wenn kein Finger auf dem Touchscreen ist.

3: Der Status des Pointers wird aktualisiert.

11.1.3 pointerMoved Methode

In dieser Methode werden die Bewegungsmethoden des `GameController` angestoßen.

```
public void pointerMoved(int pointerIndex, Vector position
```

```
            , Vector tapVector) {

      [...]

      pointer.get(pointerIndex).update(position);

/*1*/ distanceSinceLongpressStart = distanceSinceLongpressStart
                        + pointer.get(pointerIndex)
                        .getDeltaPosition().getLength();

      if (distanceSinceLongpressStart > LONGPRESS_MAX_MOVEMENT) {

            longPress.cancel();
      }

/*2*/ if (!playerObjectMovement) {

            if (pointerIndex == 0) {

/*3*/             GameController.getInstance().turnPlayer([...]);

            } else {

/*4*/             GameController.getInstance().movePlayer([...]);
            }

/*5*/ } else {

            if (pointerIndex == 0) {

/*6*/             GameController.getInstance().movePlayerObject(0
                              , tapVector);
            } else {

/*7*/             GameController.getInstance().movePlayerObject(
                              pointer.get(1).getDeltaPosition().getY()
                              / STRAFE_CORRECTION, null);
            }
      }
}
```

Code 5: Ausschnitt der Methode pointerMove aus der Klasse TabAndGo

1: Bei jeder Bewegung soll gespeichert werden wie weit sich der Finger seit dem Start eines *Longpress* bewegt hat, um den *Longpress* bei zu starker Bewegung abzubrechen.

2: Wenn sich das Programm nicht im Objektbewegungsmodus befindet, soll bei der Bewegung des ersten Fingers der Spieler rotiert werden (er schaut sich um), Code dazu bei Punkt 3. Übergeben werden die Rotationswinkel um die x- und y-Achse, welche hier allerdings ausgeblendet sind, da die Rechnung etwas unübersichtlich erscheint.
Bei Bewegung des zweiten Fingers wird der Spieler bewegt (Punkt 4), dazu wird die Bewegungslänge nach vorne bzw. hinten und zur Seite übergeben, welche ebenfalls aus Übersichtlichkeitsgründen ausgeblendet sind.

5: Befindet sich das Programm im Objektbewegungsmodus, soll mit dem ersten Finger das aktuelle Spielobjekt bewegt werden (Punkt 6). Mit dem zweiten Finger soll das Spielobjekt in der Tiefe verschoben werden (Punkt 7).

11.2 Ermittlung der Soll-Position eines Spielobjektes

In dem Interaktionskonzept TabAndGo kann ein Spielobjekt mit Hilfe der Z technique bewegt werden. Eine kurze Auffrischung was die Z technique ausmacht: Das Objekt kann mit dem Finger gewählt und parallel zur Projektionsebene verschoben werden, zudem kann es mit einem zweiten Finger durch Schiebegesten in der Tiefe bewegt werden.
Vorweg ist zu sagen, dass das Spielobjekt nicht ganz genau parallel zur Projektionsebene verschoben wird, vielmehr wird es in einer Kreisbahn um die Kamera verschoben, dies fällt allerdings kaum auf.

Beschrieben wird hier die Methode `movePlayerObject`, welche für die Umsetzung der Z technique zuständig ist.

```
public void movePlayerObject(float frontDistance, Vector tabVector) {

      [...]

/*1*/ if (tabVector != null) {
            lastTabVector = tabVector;
      }

/*2*/ float distance = Math3D.getDistance(Player.getInstance()
            .getPosition(), Player.getInstance().getObjectHoldInHand()
            .getPosition());
      distance = distance + frontDistance;

/*3*/ [...]

/*4*/ Vector theoPlayerObjectPos = lastTabVector.multCopy(distance);

/*5*/ Vector relPlayerDistanceVector = Player.getInstance()
            .getObjectHoldInHand().getPosition().subCopy(
            Player.getInstance().getPosition());
      float yRot = relPlayerDistanceVector.getAngle(Vector.Y);
      theoPlayerObjectPos.rotateSelf(-yRot, Vector.Y);
      theoPlayerObjectPos.rotateSelf(this.selectionAngleX, Vector.X);
      theoPlayerObjectPos.rotateSelf(yRot + this.selectionAngleY,
            Vector.Y);


/*6*/ [...]

/*7*/ theoPlayerObjectPos = theoPlayerObjectPos.addCopy(
                  Player.getInstance().getPosition());
      Player.getInstance().getObjectHoldInHand().prepareMovement(
                  theoPlayerObjectPos);
      Player.getInstance().getObjectHoldInHand().commitMovement();
```

```
}
```

Code 6: Die Methode movePlayerObject der Klasse GameController

Die Methode bekommt, mit dem Parameter `frontDistance` übergeben, wie weit und in welche Richtung sich das Spielobjekt in der Tiefe verschieben soll. Ist der Parameterwert negativ, soll das Spielobjekt zum Spieler verschoben werden, ansonsten von ihm weg.
Der `tabVector` gibt an, an welche Position sich das Objekt Bewegen soll.

1: Soll das Spielobjekt nach vorne verschoben werden, ist dazu der Finger zuständig, der sich nicht auf dem Objekt befindet, daher soll dessen `tabVector` nicht berücksichtigt werden, stattdessen soll der zuletzt bekannte `tabVector` des ersten Fingers verwendet werden.

2: Es wird ermittelt welche Entfernung das Spielobjekt zum Spieler hat. Danach wird zu dieser Entfernung die gewollte Tiefenbewegung (`frontDistance`) hinzugefügt.

3: Dieser Ausschnitt wurde zu Gunsten der Übersichtlichkeit entfernt. Hier wird überprüft, ob das Spielobjekt nach der Tiefenbewegung den Spieler treffen würde. Wäre dies der Fall, wird die Methode ohne das Objekt zu bewegen verlassen.

4: Hier wird die theoretische neue Position des Spielobjekts relativ zum Spieler berechnet, indem der *`tabVector`* mit der Spieler-Spielobjekt-Distanz multipliziert wird.

5: Da der Finger nicht immer auf dem Objekt platziert sein muss, sondern auch weit danebenliegen kann, muss die relative Fingerposition in Form von Winkeln der bisherigen theoretischen Spielobjekt-Position hinzugefügt werden.
Die Winkel wurden bereits bestimmt, als der erste Finger das Display berührte, dazu ist die Methode `setObjectFingerOffset` zuständig.

6: Auch dieser Abschnitt wurde zu Gunsten der Übersichtlichkeit entfernt. Überprüft wird hier, ob das Spielobjekt den Displayrand überschreiten würde, wäre dies der Fall, wird die Methode beendet, ohne das Spielobjekt zu bewegen.

7: Die relative Position des Spielobjekts zum Spieler wird hier auf die globale Position gerechnet. Danach wird das Objekt schlussendlich für die Bewegung vorbereitet, dies beinhaltet die Kollisionsabfrage. Fand keine Kollision statt, wird das Objekt an die Zielposition bewegt.

11.3 Vorbereitung der Bewegung eines Spielobjektes

Nach der theoretischen Beschreibung aus **10.2 Kollisionserkennung**, wird nun die Implementierung grob beleuchtet. Eine detailierte Beleuchtung wäre leider zu komplex und unübersichtlich.

Die Bewegung und Kollisionserkennung eines Spielobjekts wird von dem Spielobjekt selbst durchgeführt. Alle Spielobjekte erben von der Klasse `GameObject`, welche zwei öffentliche Methoden besitzt das Spielobjekt zu bewegen.

Zuerst wird mit der Methode `prepareMovement` eine Kollisionserkennung durchgeführt und berechnet wie weit sich das Spielobjekt tatsächlich bewegen darf. Mit der Methode `commitMovement` wird das Spielobjekt schlussendlich bewegt.

```
public void prepareMovement(Vector newPosition) {

      [...]

/*1*/ Vector movement = newPosition.subCopy(this.getPosition());
      LogicSphere theoNewOuterCollisionBall;
      theoNewOuterCollisionBall = this.outerCollisionBall.getCopy();
      theoNewOuterCollisionBall.getPosition().addSelf(movement);

/*2*/ List<SimplePlane> tempPossipleGrounds =
            this.detectPossipleCollisionPlanes(
            GameController.getInstance().getLevel().getGrounds(),
            theoNewOuterCollisionBall);
/*3*/ [...]


      for (LogicSphere collusionBall : collisionBalls) {

/*4*/       movement = this.checkSimplePlaneCollision(collusionBall,
                  movement, tempPossipleGrounds);

            if (movement == null) {

                  break;
            }

/*5*/       [...]

      }

      if (movement == null) {

            preparedMovement = movement;
            return;
      }

      for (SimplePlane plane : this.collidablePlanes) {

            SimplePlane tempPlane = plane.getFakeCopy();
            tempPlane.setRotatedY(tempPlane.getRotatedY() +
                  this.getRotatedY());
```

```
            tempPlane.getPosition().setTo(this.position.addCopy(
                  tempPlane.getPosition()));

            tempPlane.getPosition().rotateSelfAroundPoint(
                  this.getRotatedY(), Vector.Y, this.position);

/*6*/       movement = this.checkGameObjectCollusion(tempPlane, movement,
                  tempPossibleObjects, true);

            if (movement == null) {

                  break;
            }
      }

      preparedMovement = movement;
}
```

Code 7: Ausschnitt der Methode prepareMovement aus der Klasse GameObject

1: Zuerst wird berechnet welche relative Bewegung von der alten zur neuen Position durchgeführt werden müsste, es wird, anders gesagt, der Bewegungsvektor bestimmt.
Zudem wird eine Kopie der äußeren Kollisionskugel erzeugt, um dieser die gewollte Bewegung hinzuzufügen. Diese Kopie wird für die Methode benötigt, die auf Kollisionen testet.

2: Der nächste Schritt ist herauszufinden welche Objekte für eine Kollision in Frage kommen. Das Level beinhaltet drei relevante Listen mit kollidierbaren Elementen: Wände, Böden und Spielobjekte. Der markierte Code ruft die Methode `detectPossipleCollisionPlanes` auf, welche eine Liste mit allen Böden zurückgibt, mit denen eine Kollision theoretisch möglich wäre. Dazu wird in dieser Methode der Abstand zu den Böden ermittelt und überprüft, ob der Abstand kleiner als die Bewegungslänge ist.

3: Der Code wurde ausgeblendet, da er ähnlich ist wie in Punkt 2, er unterscheidet sich darin, dass nach kollidierbaren Wänden und Objekten gesucht wird.

4: Nun wird mit allen vorausgewählten Böden die Kollisionserkennung mit den Kollisionskugeln durchgeführt, dazu wird die temporär erzeugte Liste von Böden an die Methode `checkSimplePlaneCollision` übergeben. Die Methode gibt einen möglichen Bewegungsvektor zurück oder **`null`** wenn keine Bewegung möglich ist.
Ist bei einer der Flächen bereits keine Bewegung mehr möglich, wird die Schleife unterbrochen und die mögliche Bewegung auf **`null`** gesetzt.

5: Hier geschieht die Kollisionsabfrage mit den Wänden und Spielobjekten, ähnlich wie in Punkt 4.

6: Als letztes wird eine Kollisionsabfrage der Kollisionsflächen mit anderen Spielobjekten durchgeführt. Dazu muss zuvor die tatsächliche Position und Rotation der Fläche berechnet werden, da nur die relative Position und Lage bekannt ist.
Auch hier gibt die Methode `checkGameObjectCollusion` die mögliche Bewegung oder `null` zurück.
Der letzte Schritt ist es die mögliche Bewegung zu speichern, welche mit der `commitMovement` Methode ausgeführt werden kann.

11.4 Realisierung der Spielaktionen

In diesem Abschnitt wird erklärt wie die Spielaktionen Implementiert wurden. Die Modellierung und Definition der Spielaktionen ist dem Abschnitt **9.2.2 Modellierung der Spielaktionen** zu entnehmen.

11.4.1 Initialisierung der Interaktionen

Als erstes wird die Initialisierung der Interaktionen erläutert, die in der Klasse `Level` geschieht.

```
/*1*/ Interaction openDoorInteraction = new Interaction();
/*2*/ openDoorInteraction.addAction(new ActionRotate(-60, 0.2f));

      /* Schlüssel */
      tempGameObject = new Key(new Vector(2, 0.25f, -4), this);
/*3*/ openDoorInteraction.setCollisionBall1(tempGameObject
            .getInteractionCollisionBall());
/*4*/ tempGameObject.addCollideListener(openDoorInteraction);
      objects.add(tempGameObject);

      /* Tür */
      tempGameObject = new Door(new Vector(3), this, resources);
      openDoorInteraction.setCollisionBall2(tempGameObject
            .getInteractionCollisionBall());
      openDoorInteraction.setActionObject(tempGameObject);
/*5*/ tempGameObject.addCollideListener(openDoorInteraction);
      objects.add(tempGameObject);
```

Code 8: Ausschnitt des Konstruktors der Klasse LevelA

1: Es wird eine Interaktion erstellt, die dazu genutzt werden soll eine Tür zu öffnen.

2: Zum öffnen der Tür, muss diese rotiert werden, daher wird der Interaktion eine `Action` hinzugefügt, die die Tür um -60° in 0,2er Schritten rotieren soll.

3: Nachdem ein Schlüssel (Instanz der Klasse `Key`) erstellt wurde, wird der `InteractionCollisionBall` des Schlüssels der Interaktion hinzugefügt, um später überprüfen zu können, ob die beiden verbundenen Objekte an den richtigen Stellen kollidiert sind.

4: Dem Schlüssel wird die `Interaction` als `CollideListener` hinzugefügt, damit der Schlüssel die Interaktion bei einer Kollision über diese informieren kann.

5: Nachdem Punkt 3 und 4 mit der Tür (Instanz der Klasse `Door`) durchgeführt wurden, wird der `Interaction` nun die Tür als betroffenes Objekt mit der Methode `setActionObject` übergeben. Das bedeutet, dass alle `Actions` der `Interaction` mit der Tür durchgeführt werden.

11.4.2 Auslösen der Interaktionen

Die Interaktionen können ausgelöst werden, indem zwei logisch miteinander verbundene Spielobjekte an der richtigen Stelle (mit den verknüpften Kollisionskugeln) miteinander kollidieren.
Während der Kollisionsabfrage zwischen zwei Spielobjekten wird gespeichert an welchen Kollisionskugeln sie sich treffen. Die dazugehörende Implementierung wird nicht gezeigt, da sie zum Umfangreich für eine solche Beschreibung wäre.
Stattdessen wird der Ausschnitt gezeigt, der Nach jeder Kollision ausgeführt wird, welche die Interaktionen auslösen kann.

```
      for (CollideListener listener : collideListener) {

            listener.onCollided(this.lastCollisionBallColidedWith
                  , preparedCollisionObject.lastCollisionBallColidedWith);
      }
```

Code 9: Ausschnitt der Methode informCollideListener der Klasse GameObject

Zu sehen ist in **Code 9**, dass das `GameObject` alle angemeldeten `CollideListener` (implementiert von `Interaction`) über eine Kollision informiert und dabei die eigene sowie die Kollisionskugel des anderen `GameObjects` übergibt.

```
      public void onCollided(LogicSphere collisionBall1
                  , LogicSphere collisionBall2) {

            if (activated) {
```

```
                return;
            }

            if ((this.collisionBall1 == collisionBall1
                        && this.collisionBall2 == collisionBall2)
                        || (this.collisionBall1 == collisionBall2
                        && this.collisionBall2 == collisionBall1)) {

                activated = true;
                startNextAction();
            }
        }
```

Code 10: Die Methode onCollide des Interface CollideListener implementiert durch die Klasse Interaction

In **Code 10** ist zu sehen, dass überprüft wird, ob die übergebenen Kollisionskugeln die Kollisionskugeln sind, die bei der Initialisierung übergeben wurden. Sind sie es nicht, wird die Methode ohne weitere Aktionen beendet.
Sind es die gespeicherten Kollisionskugeln, wird die Interaktion als aktiv gekennzeichnet, sodass sie nicht erneut gestartet werden kann. Zudem wird die Methode `startNextAction` aufgerufen, wodurch die Interaktion schlussendlich ausgelöst wird.

11.4.3 Ausführen der Aktionen

Wurde die Interaktion ausgelöst, müssen alle dazugehörigen Aktionen nacheinander ausgeführt werden, dies wird durch die Methode `startNextAction` angestoßen, welche in **Code 11** gezeigt wird.

```
        private void startNextAction() {

            if (actions.size() > currentAction) {

                actions.get(currentAction).initialize(actionObject);
                currentAction++;
            }

        }
```

Code 11: Die Methode startNextAction der Klasse Interaction

In der Klasse `Interaction` wird in der Member-Variable `currentAction` gespeichert, welche Aktion als nächstes ausgeführt werden soll. Diese Aktion wird in der Methode `startNextAction` initialisiert, dazu wird ihr mit der Methode `initialize` das Spielobjekt übergeben, das bei der Initialisierung für die Aktionen bestimmt wurde.

Die Methode `initialize` ist im folgenden Codeabschnitt **Code 13** zu betrachten.

```
        public void initialize(GameObject object) {
```

```
            this.object = object;
            GameController.getInstance().addAction(this);
      }
```

Bei der Initialisierung einer Aktion übergibt diese sich selbst an dem `GameController`, welcher die Einzelschritte anstoßen wird. Zudem speichert die `Action` das zu verwendende `GameObject`, um die Aktionen darauf anzuwenden.

Der nächste Schritt ist das tatsächliche Ausführen der Einzelschritte, was im `update Timer` des `GameController` passiert. Der `update Timer` führt noch einige andere Aufgaben durch, diese wurden hier aus Übersichtsgründen ausgeblendet.

Code 12: Die Methode run des Timers update in der Klasse GameController

```
      @Override
      public void run() {
            [...]
/*1*/       mutexActiveActions.lock();
            try {

                  boolean isActionToDelete = false;
                  int i;
                  int max = activeActions.size();

/*2*/               for (i = 0; i < max; i++ ) {
                        if (!activeActions.get(i).isDone()) {

                              activeActions.get(i).doActionStep();
                        }

                        if (activeActions.get(i).isDone()) {
                              isActionToDelete = true;
                        }
                  }

/*3*/               if (isActionToDelete) {

                        deleteLastFinishedActions();
                  }
            } finally {
                  mutexActiveActions.unlock();
            }
            [...]
      }
```

Code 13: Die Methode initialize der Klasse Action

1: An dieser Stelle wird ein Mutex gelockt, um zu verhindern, dass während des Lesens der `activeActions`-Liste dieser weitere Einträge hinzugefügt werden. Würde die Liste nicht vor Zugriffen anderer Threads geschützt werden, könnte dies zu einem Fehler führen.

2: Die Schleife arbeitet alle Aktionen ab, die noch nicht als erledigt gekennzeichnet wurden. Dazu wird die Methode `doActionStep` aufgerufen, welche einen Einzelschritt der Aktion durchführt.

3: Ist eine der Aktionen in der Liste abgeschlossen, soll diese aus der Liste mit der Methode `deleteLastFinishedActions` entfernet werden.

Ein Beispiel der Methode `doActionStep` ist in zu **Code 14** sehen:

```
      @Override
      public void doActionStep() {

/*1*/       if ([...]) {

/*2*/             if (this.object.rotateY(angleStep)) {

                        angleReached += angleStep;
                  }
            } else {

/*3*/             this.actionDone();
            }
      }
```

Code 14: Die Methode doActionStep der Klasse ActionRotated

1: Hier wurde die Bedingung entfernt, da sie unübersichtlich wirkt. Es wird lediglich abgefragt, ob der gewünschte Winkel erreicht ist.

2: An dieser Stelle wird das Spielobjekt rotiert. Die Rotationsfunktion des Spielobjekts gibt zurück, ob die Teilrotation möglich war, oder ob eine Kollision stattgefunden hat.
Konnte die Rotation durchgeführt werden, kann dieser Schritt gespeichert werden, um beim nächsten Aufruf ermitteln zu können, ob die Zielrotation erreicht ist.

3: Wurde der Zielwinkel erreicht, kann die Aktion als beendet gekennzeichnet werden. Dies beinhaltet, dass die `Interaction` (die Eigentümer der `Action` ist) die Beendigung der Aktion informiert wird, damit sie die nächste `Action` startet, oder sich als ebenfalls beendet kennzeichnet.

12 Usertest

Um zu überprüfen welches der beiden implementierten Interaktionskonzepte eine bessere Userakzeptanz erhält, wird ein Usertest durchgeführt.

12.1 Ablauf

Der Testleiter zeigt dem Testuser an einem Testlevel die Bedienung der Konzepte, danach hat der Testuser Zeit sich ebenfalls im Testlevel an den Konzepten zu versuchen. Fühlt der er sich bereit, kann er das erste Level starten.
Zu erwähnen ist noch, dass die Konzepte in abwechselnder Reihenfolge getestet werden, da die Reihenfolge Einfluss auf die anschließende Bewertung hat.

Wurden beide Konzepte getestet, wird dem Testuser ein Fragebogen vorgelegt, welcher hauptsächlich fragen nach dem **3.5.2.3 Semantic differential scale** enthält.

Ist der Usertest abgeschlossen, werden die Fragebögen mit dem Statistikprogramm *R* ausgewertet.

12.2 Fragebogen

Fragebogen zum Usertest des InteractionGames

vPad und TabAndGo im Vergleich

	TabAndGo	vPad
Für wie intuitiv empfanden Sie die Gesten?	sehr ☐☐☐☐☐☐ wenig	sehr ☐☐☐☐☐☐ wenig
Wie fanden Sie die Methode die Kamera zu drehen und zu neigen?	gut ☐☐☐☐☐☐ schlecht	gut ☐☐☐☐☐☐ schlecht
Wie stufen Sie die Geschwindigkeit der Drehung und Neigung der Kamera ein?	zu schnell ☐☐☐☐☐☐ zu langsam	zu schnell ☐☐☐☐☐☐ zu langsam
Wie fanden Sie die Methode die Spielfigur zu bewegen?	gut ☐☐☐☐☐☐ schlecht	gut ☐☐☐☐☐☐ schlecht
Wie genau gelang Ihnen die Positionierung der Spielfigur?	sehr genau ☐☐☐☐☐☐ ungenau	sehr genau ☐☐☐☐☐☐ ungenau
Wie genau gelang ihnen die Positionierung der Objekte?	sehr genau ☐☐☐☐☐☐ ungenau	sehr genau ☐☐☐☐☐☐ ungenau
Sollten die Bewegungsachsen umgekehrt werden?	☐ ja ☐ nein	☐ ja ☐ nein
Welches Konzept hat Ihnen besser gefallen?	☐	☐

TabAndGo

Welche Geste zur Bewegung der Spielfigur war Ihnen lieber?	☐ Doppeltab auf den Boden ☐ Zieh- und Schiebegeste

vPad

Für wie störend empfanden Sie es, dass zwischen Objekt- und Kamerabewegungsmodus manuell umgeschaltet werden musste?	störend ☐☐☐☐☐☐ nicht störend

Sonstige Spielmechanik

Wie empfanden Sie die Selektion der Objekte?	natürlich ☐☐☐☐☐☐ unnatürlich
Wie fanden Sie die Idee, dass die Objekte direkt mit Ihrer Umgebung interagieren mussten?	gut ☐☐☐☐☐☐ schlecht
Empfanden Sie die Vibrationen bei Kollisionen als hilfreich?	hilfreich ☐☐☐☐☐☐ störend

Allgemein

Alter	☐ 16-20 ☐ 21-25 ☐ 26-30 ☐ 31-40 ☐ über 40
Geschlecht	☐ m ☐ w
Für wie geschickt stufen Sie sich bei der Steuerung virtueller Figuren ein (z.B. in PC Spielen)?	geschickt ☐☐☐☐☐☐ ungeschickt
Kannten Sie das Bewegungskonzept des vPads bereits aus anderen Spielen für Smartphones?	☐ ja ☐ nein
Mit welchem Konzept haben Sie begonnen?	☐ TabAndGo ☐ vPad

Abbildung 23: Fragebogen zum Usertest

Der erste Abschnitt des Fragebongens vergleicht einige Interaktions-Methoden direkt miteinander, dadurch soll bei der Auswertung beispielsweise ermittelt werden welche der Methoden eine genauere Steuerung zugelassen hat.

Im darauf folgenden Bereich gibt es einzelne Fragen zu den Konzepten, um über ihre nicht vergleichbaren Eigenschaften Aussagen machen zu können.

Der Testuser wird auch über Spielmechanik befragt, um bewerten zu können, ob an der Testumgebung Verbesserungen vorgenommen werden können.

Darauf folgt der allgemeine Teil, in dem der Tester Angaben über seine Person machen kann.

Auf der Rückseite des Blattes ist die Möglichkeit gegeben Kommentare zu schreiben.

12.3 Allgemeine Daten zum Usertest

Insgesamt haben 23 Personen an dem Test teilgenommen. Der Test wurde an der Hochschule RheinMain durchgeführt. Alle Tester waren Informatiker bzw. angehende Informatiker. 13% der Testuser waren weiblich, 87% männlich.

Altersverteilung: 21-25 Jahre 60%
26-30 Jahre 30%
31-40 Jahre 10%

52% der Tester kannten das Konzept des vPads bereits aus anderen Spielen für Smartphones.

Die meisten Tester stuften sich für geschickt im Umgang mit der Steuerung virtueller Figuren ein, wie man dem Boxplot aus **Abbildung 24** entnehmen kann. Die Wertung 1 ist mit "geschickt" gleichzusetzen, die Wertung 7 mit "ungeschickt".

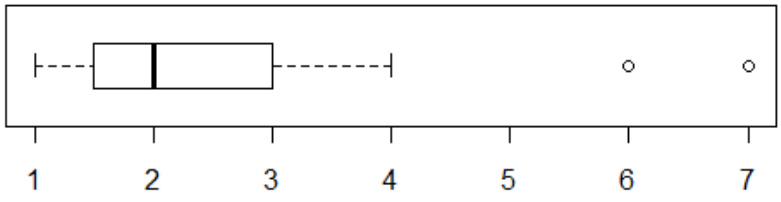

Abbildung 24: Boxplot zur Geschicklichkeitsangabe

12.4 Auswertung der direkten Gegenüberstellung der Konzepte

In diesem Abschnitt werden die Daten ausgewertet, die durch die direkte Gegenüberstellung der Konzepte gewonnen wurden.
Die folgenden Hypothesen wurden mit dem Wilcoxon-Mann-Whitney Test überprüft.

12.4.1 Intuition der Gesten

Frage

Für wie intuitiv empfanden Sie die Gesten?

Verteilung der Daten

Wertedefinition des in **Abbildung 25** zu sehenden Boxplots: 1 = sehr, 7 = wenig

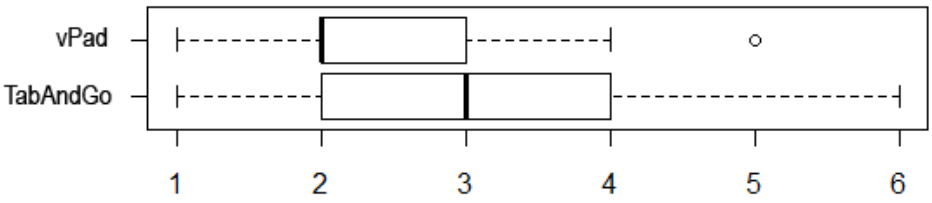

Abbildung 25: Boxplot der Daten zur Intuition der Gesten

Abzulesen ist, dass mehr Tester vPad bei dieser Frage besser bewertet haben als TabAndGo, daher wird die Hypothese zu Gunsten des vPads aufgestellt.

Hypothese

Die Gesten des vPads sind intuitiver als die von TabAndGo.

Ergebnis

```
Wilcoxon signed rank test with continuity correction

data:  data$Intuitiv2 and data$Intuitiv1
V = 42, p-value = 0.02886
alternative hypothesis: true location shift is less than 0
```

Die Auswertung mit R betätigt die Hypothese.

12.4.2 Orientierung der Kamera

Frage

Wie fanden Sie die Methode die Kamera zu drehen und zu neigen?

Verteilung der Daten

Wertedefinition des in **Abbildung 26** zu sehenden Boxplots: 1 = gut, 7 = schlecht

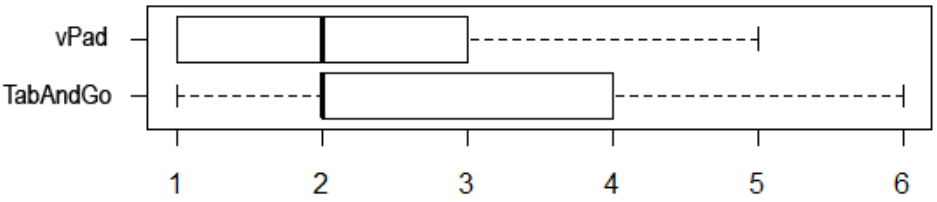

Abbildung 26: Boxplot der Daten zur Orientierung der Kamera

Hier ist nicht eindeutig welches der Konzepte besser bewertet wurde. Testweise wird die Hypothese zu Gunsten des vPads ausgelegt.

Hypothese

Die Methode die Kamera zu drehen und zu neigen ist bei vPad besser als bei TabAndGo.

Ergebnis

```
Wilcoxon signed rank test with continuity correction

data:  data$Kamera2 and data$Kamera1
V = 39.5, p-value = 0.3477
alternative hypothesis: true location shift is less than 0
```

Die Auswertung mit R kann die Hypothese nicht bestätigen, da die Irrtumswahrscheinlichkeit von 35% über der 5% Grenze liegt. Auch eine Gegenhypothese wurde durch R mit einer Irrtumswahrscheinlichkeit von 67% widerlegt.

12.4.3 Bewegung der Spielfigur

Frage

Wie fanden Sie die Methode die Spielfigur zu bewegen?

Verteilung der Daten

Wertedefinition des in **Abbildung 27** zu sehenden Boxplots: 1 = gut, 7 = schlecht

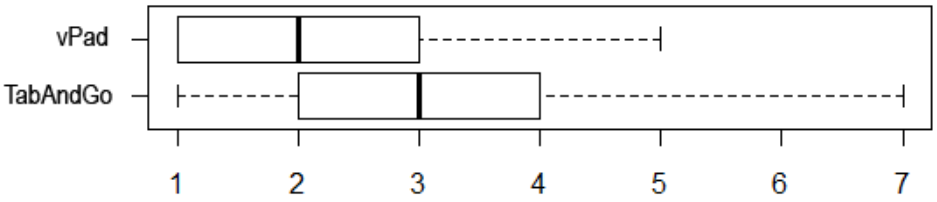

Abbildung 27: Boxplot der Daten zur Bewegung der Spielfigur

Abzulesen ist, dass mehr Tester vPad bei dieser Frage besser bewertet haben als TabAndGo, daher wird die Hypothese zu Gunsten des vPads aufgestellt.

Hypothese

Die Methode die Spielfigur zu Bewegen ist bei vPad besser als bei TabAndGo.

Ergebnis

```
Wilcoxon signed rank test with continuity correction

data:  data$BewFigur2 and data$BewFigur1
V = 50.5, p-value = 0.02004
alternative hypothesis: true location shift is less than 0
```

Die Auswertung mit R betätigt die Hypothese.

12.4.4 Positionierungsgenauigkeit der Spielfigur

Frage

Wie genau gelang Ihnen die Positionierung der Spielfigur?

Verteilung der Daten

Wertedefinition des in **Abbildung 28** zu sehenden Boxplots: 1 = sehr genau, 7 = ungenau

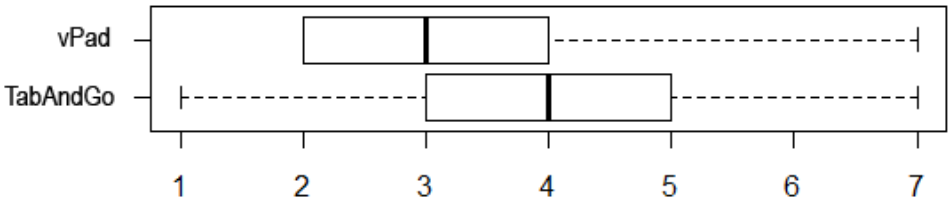

Abbildung 28: Boxplot der Daten zur Positionierung der Spielfigur

Abzulesen ist, dass mehr Tester vPad bei dieser Frage besser bewertet haben als TabAndGo, daher wird die Hypothese zu Gunsten des vPads aufgestellt.

Hypothese

Die Positionierung der Spielfigur gelingt mit vPad genauer als mit TabAndGo.

Ergebnis

```
Wilcoxon signed rank test with continuity correction

data:  data$PosFigur2 and data$PosFigur1
V = 40, p-value = 0.007223
alternative hypothesis: true location shift is less than 0
```

Die Auswertung mit R betätigt die Hypothese.

12.4.5 Positionierungsgenauigkeit der Objekte

Frage

Wie genau gelang ihnen die Positionierung der Objekte?

Verteilung der Daten

Wertedefinition des in **Abbildung 29** zu sehenden Boxplots: 1 = sehr genau, 7 = ungenau

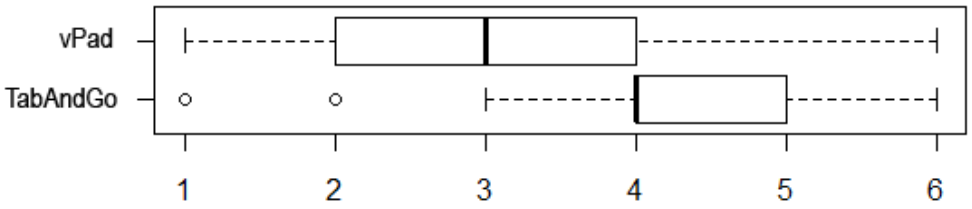

Abbildung 29: Boxplot der Daten zur Positionierung der Objekte

Abzulesen ist, dass mehr Tester vPad bei dieser Frage besser bewertet haben als TabAndGo, daher wird die Hypothese zu Gunsten des vPads aufgestellt.

Hypothese

Die Positionierung gelingt mit vPad besser als mit TabAndGo.

Ergebnis

```
Wilcoxon signed rank test with continuity correction

data:  data$PosObjekte2 and data$PosObjekte1
V = 4, p-value = 0.001844
alternative hypothesis: true location shift is less than 0
```

Die Auswertung mit R betätigt die Hypothese.

12.4.6 Geschwindigkeit von Drehung und Neigung der Kamera

Frage

Wie stufen Sie die Geschwindigkeit der Drehung und Neigung der Kamera ein?

Verteilung der Daten

Wertedefinition des in **Abbildung 30** zu sehenden Boxplots: 1 = zu schnell, 7 = zu langsam

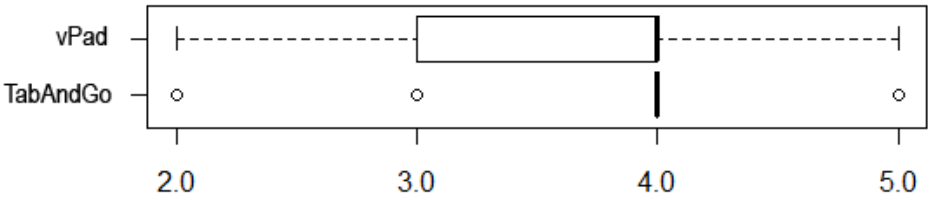

Abbildung 30: Boxplot der Daten zur Beurteilung der Dreh- und Neigegeschwindigkeit

Zu erkennen ist, dass bei TabAndGo viele Tester mit der Geschwindigkeit zufrieden waren. Bei vPad waren die meisten auch zufrieden mit der Geschwindigkeit, die abweichende Tendenz geht dahin, dass die Bewegung zu schnell war.

Hypothese

Die Geschwindigkeit der Drehung und Neigung der Kamera ist bei TabAndGo angenehmer als bei vPad.

Umstrukturierung der Daten

Die Daten wurden dahingehen umstrukturiert, dass der Wert 4 als "angenehm" angenommen wurde und als neuer Wert 1 gilt, da er nicht zu schnell und auch nicht zu langsam ist.
Die Werte 3 und 5 gelten nun als Wert 2. Die Werte 2 und 6 gelten als Wert 3. Die Werte 1 und 7 sind nun als Wert 4 gültig.

Ergebnis

```
Wilcoxon signed rank test with continuity correction

data:  data$Geschw1 and data$Geschw2
V = 18, p-value = 0.1654
alternative hypothesis: true location shift is less than 0
```

Die Hypothese kann nicht bestätigt werden, da die Irrtumswahrscheinlichkeit mit 16% zu hoch ist. Auch die Gegenhypothese, dass die Bewegungsgeschwindigkeit beim vPad angenehmer war, konnte ebenfalls nicht bestätigt werden.

12.5 Auswertung der Daten zu vPad

Frage

Für wie störend empfanden Sie es, dass zwischen Objekt- und Kamerabewegungsmodus manuell umgeschaltet werden musste?

Verteilung der Daten

Wertedefinition des in **Abbildung 31** zu sehenden Boxplots: 1 = störend, 7 = nicht störend

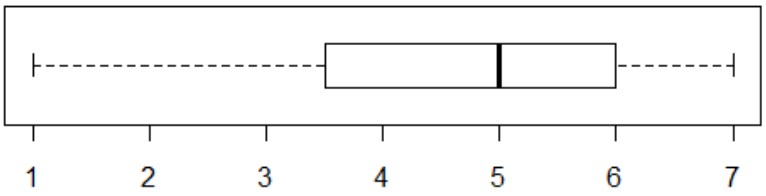

Abbildung 31: Boxplot der Daten zum Moduswechsel

Zu erkennen ist, dass die Benutzer zu "nicht störend" tendieren.

Hypothese

Der Wechsel zwischen den Bewegungsmodi wird nicht als störend empfunden (die Werte 5-7 gelten als nicht störend, der Rest als störend).

Ergebnis

```
Exact binomial test

data:  sum(data$v9 > 4) and length(data$v9)
number of successes = 13, number of trials = 23, p-value = 0.3388
alternative hypothesis: true probability of success is greater than 0.5
95 percent confidence interval:
 0.3753936 1.0000000
sample estimates:
probability of success
             0.5652174
```

Die Hypothese kann nicht bestätigt werden, da die Irrtumswahrscheinlichkeit mit 34% zu hoch ist. Auch die Gegenhypothese konnte nicht bestätigt werden.

12.6 Auswertung der Daten zu TabAndGo

Frage

Welche Geste zur Bewegung der Spielfigur war Ihnen lieber?

Verteilung der Daten

Wertedefinition des in **Abbildung 32** zu sehenden Boxplots: 1 = Doubletap auf den Boden, 2 = Zieh- und Schiebegeste

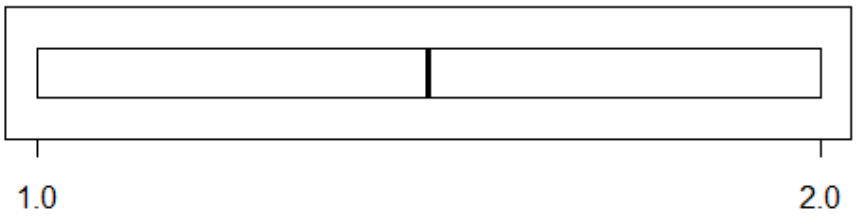

Abbildung 32: Boxplot der Daten zur favorisierten Geste

Zu erkennen ist, dass keine starke Tendenz zu einer der beiden Gesten besteht.

Hypothese

Doppeltab ist beliebter als die Zieh- und Schiebegeste.

Ergebnis

```
Exact binomial test

data:  sum(data$t9 < 2) and length(data$t9)
number of successes = 11, number of trials = 22, p-value = 0.5841
alternative hypothesis: true probability of success is greater than 0.5
95 percent confidence interval:
 0.3112636 1.0000000
sample estimates:
probability of success
                   0.5
```

Die Hypothese wird durch den Test nicht bestätigt, auch die Gegenhypothese kann durch einen Test nicht bestätigt werden.

Zu erwähnen ist, dass beide Möglichkeiten zu je 50% ausgewählt wurden.

12.7 Auswertung der Angaben zur Spielmechanik

In folgenden Auswertungen wurden die Werte 1-3 als positiv und 4-7 als negativ gewertet. Die Auswertungen wurden mit dem Vorzeichentest durchgeführt.

12.7.1 Objektselektion

Frage

Wie empfanden Sie die Selektion der Objekte?

Verteilung der Daten

Wertedefinition des in **Abbildung 33** zu sehenden Boxplots: 1 = natürlich, 7 = unnatürlich

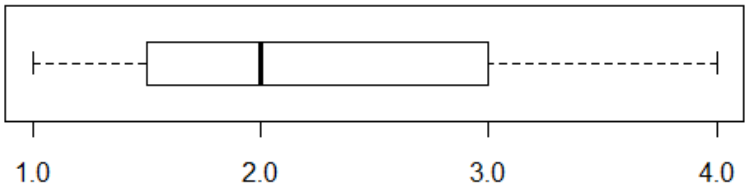

Abbildung 33: Boxplot der Daten zur Selektion der Objekte

Zu erkennen ist, dass die Tendenz dahin geht, dass die Selektion der Objekte natürlich ist. Aufgrund dessen wird die Hypothese positiv ausgelegt.

Hypothese

Die Selektion der Objekte wird als natürlich empfunden.

Ergebnis

```
Exact binomial test

data:  sum(data$s1 < 4) and length(data$s1)
number of successes = 18, number of trials = 23, p-value = 0.005311
alternative hypothesis: true probability of success is greater than 0.5
95 percent confidence interval:
 0.5961011 1.0000000
sample estimates:
probability of success
             0.7826087
```

Die Auswertung mit R betätigt die Hypothese.

12.7.2 Idee der Objektinteraktion

Frage

Wie fanden Sie die Idee, dass die Objekte direkt mit Ihrer Umgebung interagieren mussten?

Verteilung der Daten

Wertedefinition des in **Abbildung 34** zu sehenden Boxplots: 1 = gut, 7 = schlecht

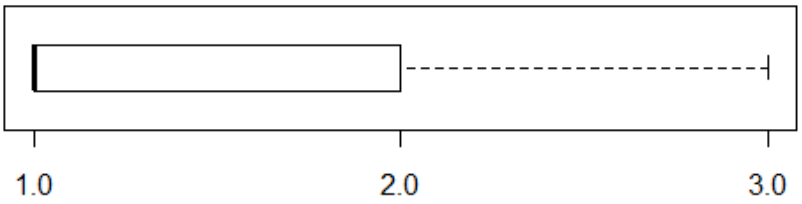

Abbildung 34: Boxplot der Daten zur Beurteilung der Idee

Abzulesen ist, dass die meisten Tester die Idee als gut empfanden. Daher wird die Hypothese positiv zur Idee ausgelegt.

Hypothese

Die Idee, dass Objekte direkt mir der Umgebung interagieren mussten, war gut.

Ergebnis

```
Exact binomial test

data:  sum(data$s2 < 4) and length(data$s2)
number of successes = 23, number of trials = 23, p-value = 1.192e-07
alternative hypothesis: true probability of success is greater than 0.5
95 percent confidence interval:
 0.8778766 1.0000000
sample estimates:
probability of success
                     1
```

Die Auswertung mit R betätigt die Hypothese.

12.7.3 Vibrationen bei Kollisionen

Frage

Empfanden Sie die Vibrationen bei Kollisionen als hilfreich?

Verteilung der Daten

Wertedefinition des in **Abbildung 35** zu sehenden Boxplots: 1 = hilfreich, 7 störend

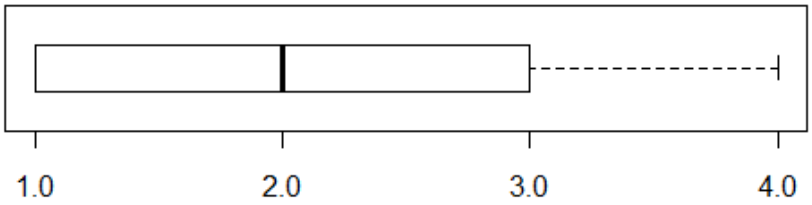

Abbildung 35: Boxplot der Daten zur Vibration bei Kollisionen

Auch hier ist abzulesen, dass die Tendenz dahin geht, dass die Vibration bei Kollisionen hilfreich ist. Aufgrund dessen wird die Hypothese positiv zur Vibration ausgelegt.

Hypothese

Die Vibration bei Kollisionen ist hilfreich.

Ergebnis

```
Exact binomial test

data:  sum(data$s3 < 4) and length(data$s3)
number of successes = 20, number of trials = 22, p-value = 6.056e-05
alternative hypothesis: true probability of success is greater than 0.5
95 percent confidence interval:
 0.7405335 1.0000000
sample estimates:
probability of success
             0.9090909
```

Die Auswertung mit R betätigt die Hypothese.

12.8 Kritiken der Tester

Schriftlich und mündlich wurde darauf hingewiesen, dass eine visuelle Kennzeichnung des aktuellen Bewegungsmodus bei vPad wünschenswert ist.

Ein Tester hat sich gewünscht, dass sich die Länge der Vibration bei der Selektion der Objekte und Kollisionen unterscheidet.

Schriftlich und mündlich wurde beanstandet, dass auswählbare Objekte nicht gekennzeichnet sind.

Ein Tester bemängelte, dass die Selektion von kleinen oder weit entfernten Objekten schwierig ist.

12.9 Fazit des Usertests

Der Usertest zeigt, dass das Konzept vPad bei den Nutzern eine bessere Akzeptanz erzielt. Es ist in fast allen Punkten beliebter als TabAndGo. Lediglich bei der Dreh- und Neigungsmethode der Kamera waren sich die Tester uneinig.
Auch bei der Frage welches Konzept die Tester besser fanden entschieden sich 74% für vPad.

Denkbar wären noch einige Verbesserungen des vPad Konzepts, beispielsweise, dass der Objektbewegungsmodus vom Kamerabewegungsmodus darin unterschieden wird, dass sich die Farbe der Steuerknüppel ändert.
Zudem könnten auch Teile der Konzepte kombiniert werden. Die Methode zur Kameradrehung und -neigung aus TabAndGo könnte zusätzlich in vPad als optionale Steuerungsmöglichkeit eingefügt werden, da diese Methode im Test nicht unbeliebt war. Allerdings sollte dann die Möglichkeit gegeben sein die Bewegungsachsen umzukehren, da ca. 40% der Testuser dies gewünscht hatten.
Auch das einfügen des Doubletaps auf den Boden zur Fortbewegung wäre denkbar.
Zudem sollten einige Bewegungsattribute einstellbar gemacht werden. Der Benutzer sollte wählen können wie schnell die Kamera gedreht und geneigt wird, da bei dem Usertest eine Uneinigkeit im Bezug auf die Geschwindigkeit bestand.

Die Testumgebung könnte ebenfalls verbessert werden, indem selektierbare Objekte visuell gekennzeichnet werden.

13 Evaluierung der Arbeit

Während der Recherche zur Arbeit konnte beobachtet werden, dass es einige Interaktionskonzepte gibt, die sich relativ stark in Form ihrer Gesten voneinander unterscheiden. Gemeinsam hatten sie allerdings, dass sie allein durch freie Gesten gesteuert wurden und auf virtuelle Steuerelemente verzichten. Nur in 3D-Spielen ähnelten sich die Interaktionskonzepte in der Form, dass eine virtuelle Steuerkonsole zur Steuerung der Spielfigur eingeblendet wurde. Zu erwähnen ist noch, dass sich, im Gegensatz zu den anderen Konzepten, keines der untersuchten Spielen mit der Bewegung von Objekten beschäftigte.

Daher wurden zwei Konzepte erarbeitet, die sich auf die gleiche Weise unterscheiden, wie die eben erwähnten Konzepte mit freien Gesten und den Spielen mit virtuellen Steuerelementen. Dieser Unterschied wurde gewählt, um überprüfen zu können, welche Methode eine höhere Benutzerakzeptanz erzielt. Erarbeitet wurde zum einen das Konzept TabAndGo, welches auf virtuelle Steuerelemente verzichtet, zum anderen das vPad, das komplett auf solche Steuerelemente vertraut.

Diese beiden Konzepte wurden durch einen Usertest miteinander verglichen, wobei sich herausstellte, dass die virtuellen Steuerelemente insgesamt beliebter waren, als freie Gesten. Allerdings waren auch Teile der freien Gesten nicht unbeliebt.
Zusammenfassen könnte man sagen, dass die Akzeptanz der Konzepte bei virtuellen Steuerelement höher ist, man allerdings auch freie Gesten einstreuen könnte. In welcher Form diese Konzeptideen gemischt werden können, müsste in einer weiteren Arbeit geklärt werden.

14 Zusammenfassung

Im Rahmen dieser Bachelor Thesis wurden vorhandene Interaktionskonzepte zur Interaktion im dreidimensionalen Raum vorgestellt und beurteilt, um die Ist-Situation besser erfassen zu können. Zudem wurden einige Teile der Konzepte zur eigenen Verwendung vorgemerkt.
Nach dieser Analyse wurden drei Konzepte erarbeitet, darunter die zwei Konzepte, die zur Implementierung ausgewählt wurden. Zum einen TabAndGo, welches sich dadurch auszeichnet auf virtuelle Steuerelemente wie Steuerknüppel zu verzichten und stattdessen nur auf Gesten reagiert. Das andere Konzept namens vPad verwendet hingegen ausschließlich virtuelle Steuerelemente, um die Interaktionen durchzuführen. Die Konzepte sind bewusst sehr gegensätzlich erstellt, um im späteren Verlauf die Akzeptanzrichtung besser bestimmen zu können.
Für die Konzepte wurde eine Testumgebung in Form eines Spiels erstellt, in dem man Objekte bewegen und mit anderen Objekten kollidieren muss, um zu gewinnen.
Auf dieser Testumgebung wurde ein Usertest an der Hochschule RheinMain durchgeführt. Die Testuser durften beide Konzepte testen und anschließend mit Hilfe eines Fragebogens bewerten. Die dabei entstandenen Daten wurden mit dem Statistikprogramm R ausgewertet und ergaben, dass das vPad beliebter war als TabAndGo.
Für weitergehende Verbesserungen in diesem Bereich, bietet sich das vPad zum Ausbau an.

15 Glossar

Aktivitätsdiagramm	"Aktivitätsdiagramme sind Diagramme zur Flussmodellierung. Sie stellen die Aktivitäten eines Systems dar, die Aktionen, aus den die Aktivitäten sich zusammensetzen und den Fluss durch die Aktivitäten. Mit Aktivitätsdiagrammen können komplexe Abläufe in einem System modelliert werden (Geschäftsprozesse, Workflows). Da Aktivitäten aus Aktionen und deren zeitlicher Verknüpfung bestehen, können sie auch zur Modellierung der internen Logik komplexer Operationen verwendet werden und somit Algorithmen visualisieren. Aktivitätsdiagramme können in Verantwortungsbereiche gegliedert werden. Damit können die Aktionen bestimmten Modellelementen, wie Klassen oder Komponenten zugeordnet werden." **[Hoc112]**
Doubletap	Eine Touchgeste, die sich durch doppeltes Antippen des Displays auf der Selben Stelle definiert.
Immersion	"Eine virtuelle Realität ist immersiv, wenn sie das "Eintauchen" eines Benutzers in die scheinbare Realität erleichtert, indem sie Reize (wie Bilder, Töne) erzeugt, die unsere menschlichen Sensoren (wie Augen, Ohren) ähnlich der Realität ansprechen." **[Dör09]**
Klassendiagramm	"Klassendiagramme stellen die statische Struktur eines Systems dar. Sie zeigen die Klassen, die Eigenschaften der Klassen (Attribute), das Verhalten (Operationen) der Klassen und die Beziehungen zwischen den Klassen. Sie sind der zentrale Diagrammtyp der UML und werden in allen Phasen der Softwareentwicklung eingesetzt." **[Hoc11]**
Longpress	Eine Touchgeste, die sich durch eine langes Gedrückthalten eines Fingers auf dem Display definiert.
Paketdiagramm	"Pakete werden verwendet um Mengen von Modellelementen zu Gruppen zusammenzufassen. Sie dienen der Strukturierung von UML-Modellen in überschaubare Einheiten. Pakete definieren einen Namensraum. Paketdiagramme stellen die Pakete und die Beziehungen (Abhängigkeiten) zwischen den Paketen eines Modells dar." **[Hoc111]**
R	"R ist eine freie Programmiersprache für statistisches Rechnen und

	statistische Grafiken. Sie ist in Anlehnung an die Programmiersprache S entstanden und weitgehend mit dieser kompatibel." **[Wik]**
Use Case Diagramm	"In Use Case Diagrammen wird das externe Systemverhalten aus Anwendersicht beschrieben. Use Case Diagramme stellen das geplante System, die Akteure, die Verwendung des geplanten Systems (Anwendungsfälle) und die Beziehungen zwischen Akteuren und Anwendungsfällen dar. Use Case Diagramme geben Auskunft darüber, was ein geplantes System aus Sichtweise der Benutzer leisten soll." **[Hoc113]**

16 Literaturverzeichnis

Bücher und elektronische Quellen

[Dör09] Dörner, Ralf: Skript zur Veranstaltung Virtual Reality Systeme (Wiesbaden, 2009).

[Jul10] Horn, Julian: Dynamisches Portieren von Android Anwendungen nach Java (Wiesbaden, 2010).

[Ste] Kleuker, Stephan: Vorlesungsfolien zur Veranstaltung Softwaretechnik (Wiesbaden, 2008).

Onlinequellen

[Ant10] Martinet, Anthony: Anthony Martinet's Home Page
[http://www.lifl.fr/~martinea/] (Zugriff: März 2010).

[Bil11] Jacobs, Bill: OpenGL tutorial
[http://www.videotutorialsrock.com] (Zugriff: April 2011).

[Bre10] Breier, Florian: mt4k.org (2010)
[mt4j.org/misc/Thesis%20Breier%20-%20Multitouch%203D.pdf] (Zugriff: März 2011).

[cne11] cnet: cnet News - Google unveils cell phone software and alliance
[http://news.cnet.com/8301-17939_109-9810937-2.html] (Zugriff: Juni 2011).

[Com] Computerbild: http://www.computerbild.de
[http://www.computerbild.de/artikel/cb-Ratgeber-Touchscreens-4563263.html] (Zugriff: April 2011).

[DGL] DGL Wiki: DGL Wiki
[http://wiki.delphigl.com/index.php/OpenGL_ES] (Zugriff: April 2011).

[Gam11] Gameloft: Gameloft - NOVA
[http://www.gameloft.de/android-spiele/nova/] (Zugriff: Juni 2011).

[Gam111] Gameloft: Gameloft - Modern Combat Sandstorm
[http://www.gameloft.de/smartphone-spiele/modern-combat-sandstorm/] (Zugriff: Juni 2011).

[Gam112] Gameloft: Gameloft - Dungeon Hunter
[http://www.gameloft.de/smartphone-spiele/dungeon-hunter/] (Zugriff: Juni 2011).

[Goo11] Google: http://developer.android.com
[http://developer.android.com] (Zugriff: März 2011).

[Goo111] Google: Android Developers - Activities
[http://developer.android.com/guide/topics/fundamentals/activities.html] (Zugriff: April 2011).

[Goo112] Google: Android Developers - 3D with OpenGL
[http://developer.android.com/guide/topics/graphics/opengl.html] (Zugriff: April 2011).

[Goo113] Google: Android Developers - Android SDK
[http://developer.android.com/sdk/index.html] (Zugriff: April 2011).

[Goo114] Google: Android Developers - Androidology - Part 1 of 3 - Architecture Overview
[http://developer.android.com/videos/index.html#v=QBGfUs9mQYY] (Zugriff: April 2011).

[Hoc11] Hochschule Darmstadt: Hochschule Darmstadt - Klassendiagramm
[http://www.fbi.h-da.de/labore/case/uml/klassendiagramm.html] (Zugriff: Juni 2011).

[Hoc111] Hochschule Darmstadt: Hochschule Darmstadt - Paketdiagramm
[http://www.fbi.h-da.de/labore/case/uml/paketdiagramm.html] (Zugriff: Juni 2011).

[Hoc112] Hochschule Darmstadt: Hochschule Darmstadt - Aktivitätsdiagramm
[http://www.fbi.h-da.de/labore/case/uml/aktivitaetsdiagramm.html] (Zugriff: Juni 2011).

[Hoc113] Hochschule Darmstadt: Hochschule Darmstadt - Use Case Diagramm
[http://www.fbi.h-da.de/labore/case/uml/anwendungsfalldiagramm.html] (Zugriff: Juni 2011).

[KHR11] KHRONOS Group: KHRONOS - OpenGL ES Overview
[http://www.khronos.org/opengles/] (Zugriff: April 2011).

[Kre11] Kreativagentur NMY: NMY
[http://www.nmy.de/portfolio/projects/3d-realtime-multi-touch-infotainment/] (Zugriff: März 2011).

[onp11] onpulson: onpulson.de
[http://www.onpulson.de/lexikon/5801/anecdotal-evidence/] (Zugriff: Mai 2011).

[ope11] open handset alliance: open handset alliance - Members
[http://www.openhandsetalliance.com/oha_members.html] (Zugriff: Juni 2011).

[tex11] texturenwelt: texturenwelt.de
[http://www.texturenwelt.de] (Zugriff: April 2011).

[The] The Matrixer: The Matrixer
[http://www.thematrixer.net] (Zugriff: April 2011).

[Wik] Wikipedia: Wikipedia - Beschleunigungssensor
[http://de.wikipedia.org/wiki/Beschleunigungssensor] (Zugriff: April 2011).

[Wik1] Wikipedia: Wikipedia - Touchscreen
[http://de.wikipedia.org/wiki/Touchscreen] (Zugriff: April 2011).

[Wik2] Wikipedia: Wikipedia - Boxplot
[http://de.wikipedia.org/wiki/Boxplot] (Zugriff: Mai 2011).

17 Anhang: Inhalt der Thesis CD

Auf der beigelegten CD befinden sich folgende Unterlagen zur Bachelor Thesis:

- Die Bachelor-Thesis als PDF
- Sämtliche Diagramme als PDF
- Das Eclipse Projekt "InteractionGame" welches den Code der Android-Anwendung zum Testen der Interaktionskonzepte, sowie die Java Dokumentation beinhaltet